A Controlled Natural Language for Business Intelligence Monitoring

Christian Colombo[1], Jean-Paul Grech[1], and Gordon J. Pace[1]

University of Malta
{christian.colombo | jean-paul.grech.11 | gordon.pace}@um.edu.mt

Abstract. With ever increasing information available in social networks, the number of businesses attempting to exploit it is on the rise, particularly by keeping track of their customers' posts and likes on social media sites like Facebook. Whilst APIs can be used to automate the tracking process, writing scripts to extract information and process it requires considerable technical skill and is thus not an option for non technical business analysts. On the other hand, off-the-shelf business intelligence solutions do not provide the desired flexibility for the specific needs of particular businesses. In this paper, we present a controlled natural language enabling non-technical users to express their queries in a language they can easily understand but which can be directly compiled into executable code.

Keywords: controlled natural languages, social networks, runtime verification

1 Introduction

Social media has provided the business community with a unique and unprecedented opportunity to engage with their customers, critics, competitors, etc. Yet, this comes at the cost of continuously monitoring various fora on which brand names may be mentioned, queries may be posted and products may be compared. Dealing effectively with social media in a context where even an hour can be considered as far too long a response time, is a challenging task.

Focusing in particular on Facebook, a typical business would have its own page as well as a strong interest in pages on which their products may be discussed or advertised. Typical events which are relevant for a business might include any mention of the brand or product, an advertising post by a competitor, a comment by a customer (particularly if negative or a question), and so on. To make the task of checking for these events manageable, dashboards [1–3] are available allowing users to specify events of interest so that a notification is received when such an event is detected (e.g., a notification when more than five comments are awaiting a response).

The problem with existing tools is that while they allow the specification of a number of events of interest, they do not offer the flexibility which might be

required for the business' specific needs. For example, one might want to prioritise the notifications in order of urgency (e.g., a comment from a new customer might be given precedence over that of an existing customer); alternatively one might want to group them into batches (e.g., a notification per five comments unless a comment has been posted for more than three hours). Such flexibility usually comes at the price of a tailor-made solution which is generally expensive both if developed in house or by a third party.

One way of allowing a high degree of flexibility while providing an off-the-shelf solution would be to present a simple interface which would allow a business intelligence analyst the flexibility to express the desired events for notification. These would in turn be automatically compiled into Facebook monitors without any further human intervention. Whilst an automated compiler would struggle to handle natural language descriptions and a non-technical business analyst would struggle with a programming language, a domain-specific language presented to the user as a controlled natural language (CNL) [7] may act as an intermediary: it provides the feel of a natural language but constraints the writer to particular keywords and patterns.

In this paper, we present a CNL (Section 2) we have developed based on the results of interviews with business analysts, supporting the expression of requests such as *'Create an alert when the service page has a post and the post contains the keywords fridge, heater, and freezer'* and *'Create an alert when my page has a post and the post is negative and the post has 10 likes'.*

The language is given an operational semantics (Section 3), which in turn enables the compilation of specifications in the language into executable monitors which can analyse traces of Facebook events. Although runtime verification [4, 8] is typically used for bug detection by matching the execution flow of a program to patterns encoded in terms of formally specified properties, runtime verification tools essentially provide specification of monitors independent of the main system. We have thus translated CNL specifications to be used by the runtime verification tool Larva [5] and then used an adapter to present relevant Facebook events as method calls in the control flow of a program.

Putting everything together, Figure 1 depicts the proposed architecture. The user — depicted on the left — writes a specification in CNL and feeds it to the CNL-to-Monitor tool. Subsequently, this tool generates two elements: a monitoring specification for the Larva monitoring tool and an event bridge which harvests Facebook events and communicates them in an appropriate form to Larva. Finally, based on the output from Larva, a dashboard is updated to give feedback to the user. The architecture we have developed can be easily adapted to new data sources (e.g. Twitter) and to alternative monitoring tools, although some work would have to go into modifying the Facebook-specific parts of the CNL.

The CNL has been evaluated (see Section 4) from the point-of-view of non-technical users through a hands-on session and follow-up questionnaire with a number of users from a small company. The results, reported herein, indicate that the users managed to understand well ready-written rules and were able

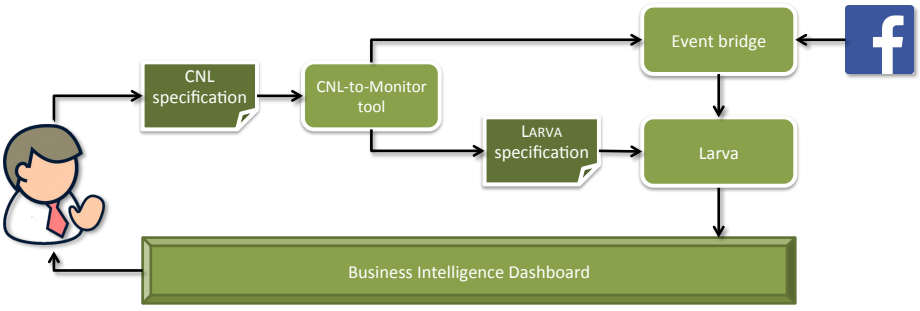

Fig. 1. The architecture of the proposed approach

to express rules using the language without any syntactic checking and user-interface support.

2 A Controlled Natural Language for Business Intelligence

The business intelligence language being proposed, based on interviews with end-users, is at core a controlled natural language [7], to enable non-technical users to experiment with different rules and modify them without the need of going through a further cycle with a developer or technical expert.

To reduce the risk of syntax errors, at the top level, the language is largely a template-based one [6] allowing for the definition of alerts as in, for example, the declaration:

> *Create an alert when the service page p has a multiple of 5 check-ins, with priority 3.*

This alert notifies the user whenever a particular page hits 5, 10, 15, etc. check-ins[1]. The priority identifies the severity of the alert, allowing us to have a tiered approach to alert handling.

Alerts can be triggered on three main types of events: (*i*) page-centric events; (*ii*) post-centric events; and (*iii*) message-centric events. All these events fire when a particular change of state happens. For instance, in the case of page-centric events, three types of event firings can be identified as shown in the following grammar fragment:

$$\begin{aligned}\langle PageEvent\rangle ::= &\ \langle Page\rangle \text{ has a post } \langle Filter\rangle \\ \mid &\ \langle Page\rangle \text{ has } \langle Count\rangle \text{ check-ins} \\ \mid &\ \text{poster } \langle UserPageAction\rangle\ \langle Page\rangle\end{aligned}$$

[1] Check-ins are events when users register their presence at a particular location or business premises.

Thus, page-centric events can fire when (*i*) a new post appears on a page; (*ii*) a number of check-ins occur on a page; or (*iii*) a user (poster) performs an action on a page (e.g., likes or shares a page). As can be seen, these events can be easily extended by allowing for alternative templates at this level.

However, at a finer level, our alert specification language loses its template-based flavour and allows for a slightly freer form of specification. Most events can be filtered by relevant features — for example, posts on a page can be filtered by placing a conjunction of constraints on the message or the poster, as can be seen in the following grammar fragment:

$$
\begin{array}{lll}
\langle \mathit{MessageFilter} \rangle & ::= & \text{has } \langle \mathit{Count} \rangle \ \langle \mathit{UserPageRelation} \rangle \\
& | & \text{has keywords } \langle \mathit{KeywordList} \rangle \\
& | & \text{is } \langle \mathit{TypeOfComment} \rangle \\
& | & \ldots \\
\langle \mathit{PosterFilter} \rangle & ::= & \text{poster } \langle \mathit{UserPageAction} \rangle \ \langle \mathit{Page} \rangle \\
& | & \text{has left } \langle \mathit{Count} \rangle \text{ posts on } \langle \mathit{Page} \rangle \\
& | & \text{is from } \langle \mathit{Location} \rangle \\
& | & \ldots
\end{array}
$$

Note that the filters take different forms, depending on the elements available for the event in question. While users posting comments can be filtered by their location, messages can be filtered by keywords. Of particular interest are: (*i*) filters based on linguistic analysis of the customer's post — the third option for a message filter in the grammar above is such a filter, allowing an alert to depend on whether a question was posted, or based on whether the comment was a positive or negative one by using an external sentiment analysis library; and (*ii*) filters based on temporal constraints, such as by day of the week, date ranges or time elapsed between two events.

Apart from alert definitions, the language also supports setting, and modifying parameters for the alert rules to trigger. For instance, a user may set the weight assigned to negative or positive posts on a page, to allow them firing when they exceed a particular threshold.

3 Monitoring Semantics

The CNL is given an operational semantics of the form $\sigma \xrightarrow{a} \sigma'$, where a is a timestamped Facebook event and σ is the configuration of the Facebook monitor, storing the relevant information, such as timers, the state of counters, etc. The rules for all the patterns supported by the CNL are represented by the rule CNL (see Figure 2). These semantics can be readily translated into a specification for a runtime verification tool such as LARVA.

Note that rules of the form $\sigma \xrightarrow{a} \sigma'$ assume that monitoring can take place as soon as an event happens, i.e. in synchrony with events happening on the Facebook platform. Unfortunately, this is not always feasible in the case of Facebook, since one is not allowed to subscribe to notifications of pages one does not own. To address this constraint, we adopt a polling-based approach, in which the system

regularly queries Facebook (through its API) and fetches relevant events which have gathered since the last query. For this reason, the monitor consumes events from a buffer rather than directly from Facebook as represented by rule Mon.

$$\textsc{Cnl}\ \frac{\cdots}{\sigma \xrightarrow{a} \sigma'} \qquad \textsc{Mon}\ \frac{\sigma \xrightarrow{a} \sigma'}{(\mathrm{fb}, \sigma)_{a:\mathrm{buf}} \xrightarrow{a} (\mathrm{fb}, \sigma')_{\mathrm{buf}}}$$

$$\textsc{Fb}\ \frac{\mathrm{fb} \xrightarrow{W} \mathrm{fb}'}{(\mathrm{fb}, \sigma)_{\mathrm{buf}} \xrightarrow{\mathrm{poll}} (\mathrm{fb}', \sigma)_{\mathrm{buf} +\!\!+ \mathrm{sort}(W)}}$$

Fig. 2. The semantic rules for monitoring Facebook events

Representing the progression of the state of Facebook pages is not as straightforward as it might appear, since Facebook can only be queried for events on a page-by-page basis, thus returning a set of unordered events in each case. Thus, mathematically, the trace fragment returned by a Facebook poll has to be chronologically sorted before being processed, as is shown in rule Fb.

Connecting back to the architecture presented in the first section, rules Mon and Fb are embodied in the *Event bridge* component which replays the events to the Larva monitor (which in turn embodies the Cnl rule) in the correct order. This logic has been successfully implemented and tested on two case studies. However, in this paper we focus on the language design aspect of this work and thus the next section describes how we evaluated the CNL in terms of its understandability and usability by non-technical users.

4 Evaluation

From an expressivity point of view, we ensured that the language supports the necessary aspects by interviewing business analysts. However, there were a number of interesting elements which were not easy to incorporate within the CNL without running into considerable complexities:

Social awareness It might be useful to distinguish between the people interacting with the business' online presence in terms of how much closely related they are to the business. For example, a like from a business employee or a close relative might be considered less important than that of a person with no links. Although a valid suggestion, we considered these social aspects to be outside the scope of our time-limited project.

Semantic analysis of posts Another proposal emerging from the interviews was to enable semantic analysis of posts, identifying adverts by third parties, distinguishing between positive and negative posts, questions from non-questions, etc. Whilst we have successfully managed to integrate two third

party projects[2] for these purposes, there are many other natural language techniques which can extract further useful information.

Apart from expressivity issues, our main concern for the proposed CNL was how easily a lay person would be able to understand expressions written in the CNL and subsequently how long it would take for the person to be able to write useful expressions in CNL. To this end, a questionnaire was used to interview thirteen non-technical persons from a local company. The questionnaire was split into two main parts as follows:

Understanding the CNL The participants were presented with a number of statements expressed in the proposed CNL (e.g., *'Create an alert when the competitor page (*`www.facebook.com/competitor`*) has a positive post and the poster has left posts on my page (*`www.facebook.com/mybusiness`*) before'*) and they were expected to explain the meaning of the statements in natural language without any supporting documentation. In each of the four cases, more than two-thirds of the respondents explained the statement correctly although some left out minor details in their explanations.

Expressing statements in the language The second exercise involved the opposite: given a textual description (e.g., *'You want to know when someone leaves a question on your page'*), respondents were expected to write it in terms of the CNL without any support (except the language samples in the previous exercise). This proved to be harder but around 60% of the respondents managed to produce an answer sufficiently close to be easily auto-correctable with the help of an appropriate user interface.

With these encouraging results, we feel that the presented work is a step in the right direction albeit requiring further development in order to make the approach more usable in practice.

5 Conclusions

In this paper, we have presented some initial experiments with the design of a controlled natural language to enable business analysts to customise a Business Intelligence dashboard. Although the language is rather contrived, it proved to be usable by non-technical experts, and can be used to effectively customise analysis of social media activity. Furthermore, the architecture can be easily adaptable to work of other streams of information e.g. Twitter, or to refer to other currently unhandled events from Facebook.

One way of increasing the usability of the CNL approach is by providing users with a richer interface (rather than simply a text editor) to write their business rules. By using a pull-down menu approach or a 'fridge magnet' interface [9] (in which the interface emulates fridge-magnet words, allowing the users to move

[2] `http://sourceforge.net/projects/chatscript` for question detection and `http://sentiment.vivekn.com` for sentiment analysis.

the words to compose valid sentences on the fridge), would make the process less error prone and would avoid error-handling and syntax debugging to which non-technical persons can be averse.

The interviews we held with business analysts also suggest that there are areas of great interest which the CNL barely touches. We hope to extend this in the future by integrating with more NLP tools on the one hand, and extracting more information from the social network to provide further data points. Finally, by looking into other industrial case studies and by having more non-technical users we hope to obtain more feedback to fine-tune the CNL to improve its comprehensibility and utility.

References

1. Facebook real-time updates. https://developers.facebook.com/docs/graph-api/real-time-updates, March 2014
2. Geckoboard. http://www.geckoboard.com, December 2013
3. Tableau software. http://www.tableausoftware.com, December 2013
4. Colin, S., Mariani, L.: Run-time verification. In: Model-Based Testing of Reactive Systems, LNCS, vol. 3472, pp. 525–555. Springer (2005)
5. Colombo, C., Pace, G.J., Schneider, G.: Larva — safer monitoring of real-time java programs (tool paper). In: SEFM. pp. 33–37. IEEE (2009)
6. Esser, M.W., Struss, P.: Obtaining models for test generation from natural-language-like functional specifications. In: 18th International Workshop on Principles of Diagnosis (2007)
7. Kuhn, T.: A survey and classification of controlled natural languages. Computational Linguistics 40(1), 121–170 (March 2014)
8. Leucker, M., Schallhart, C.: A brief account of runtime verification. The Journal of Logic and Algebraic Programming 78, 293 – 303 (2009)
9. Ranta, A.: Grammatical framework. J. Funct. Program. 14(2), 145–189 (2004), http://dx.doi.org/10.1017/S0956796803004738