A DSL for Business Intelligence Monitoring

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

University of Malta

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 costs of continuously monitoring the various fora where the business name may be mentioned, where questions may be posted, where products may be compared, etc. Effectively dealing with social media in a world where a maximum of a few hours is the expected 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 a product, an advertising post by a competitor, a comment by a customer (particularly if negative or if a question), and so on. To make the task of checking for these events manageable, dashboards [1, 3, 2] 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 by the business in question. For example one might want to prioritise the notifications in order of urgency (e.g., a comment from a new customer might is 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. While 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) [6] may act as an intermediary: it provides the feel of a natural language but constraints the writer to particular keywords and patterns. Furthermore, using standard techniques to allow the user to write their scripts using particular user-interfaces (e.g. pull-down menus or fridge magnet style) bypasses the problem of syntax errors, thus making their writing more accessible to a non-technical audience.

Based on interviews with two business analysts, such a CNL should allow the user to specify patterns such as: (i) *Create an alert when the service page has a post and the post contains the keywords fridge, heater, and freezer.* (ii) *Create an alert when my page has a post and the post is negative and the post has 10 likes.*

A number of the events proposed by the domain experts were significantly challenging or outright impossible unless further data is supplied:

Sentiment analysis While measuring the number of likes is easy, detecting whether a post is negative or positive is harder. To cater for this, we have used a ready-made sentiment analysis library[1] which analyses posts and reports whether the post is positive, negative, or neutral.

Distinguishing between users If employees post regularly on the business' page, then one would typically want to filter out their posts for the purposes of alert raising. Unfortunately, this functionality cannot be provided unless further information is available (e.g., a list of the employees).

Once a prototype CNL was designed, it could have potentially been compiled into any executable programming language. With the aim of keeping the translation as simple as possible, we translated the CNL into a intermediary specification from the runtime verification domain. Runtime verification [4, 7] is typically used for bug detection by matching the execution flow of a program to patterns encoded in terms of formally specified properties. Translating our CNL into the formal specification accepted by the runtime verification tool Larva [5] and using a simple adapter to present relevant Facebook events as method calls in the control flow of a program, we were able to detect Facebook behaviour through runtime verification software.

When it came to evaluating the CNL, we focused on the practical aspects of non-technical users using the CNL. To this end, a questionnaire was used to interview a number of users 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 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 these include those who left out minor details in the description.

Expressing statements in the language The second exercise involved the opposite: given a textual description, respondents were expected to write it in terms of the CNL without any support. This proved to be harder but around 60% of the respondents got a good answer or a close enough answer which would have probably been correct with the help of a graphical user interface providing syntactic checks and auto-complete functionality.

In the future, we aim to continue evaluating the CNL with more users, enabling us to fine tuning it and improve its implementation.

References

1. Facebook real-time updates. https://developers.facebook.com/docs/graph-api/real-time-updates. March 2014.

[1] `http://sentiment.vivekn.com`

2. Geckoboard. http://www.geckoboard.com. December 2013.
3. Tableau software. http://www.tableausoftware.com/. December 2013.
4. S. Colin and L. Mariani. Run-time verification. In *Model-Based Testing of Reactive Systems*, volume 3472 of *LNCS*, pages 525–555. Springer, 2005.
5. C. Colombo, G. J. Pace, and G. Schneider. Larva — safer monitoring of real-time java programs (tool paper). In *SEFM*, pages 33–37. IEEE, 2009.
6. T. Kuhn. A survey and classification of controlled natural languages. *Computational Linguistics*, 40(1):121–170, March 2014.
7. M. Leucker and C. Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78:293 – 303, 2009.