The Rise of the (Modelling) Bots: Towards Assisted Modelling via Social Networks

Sara Pérez-Soler, Esther Guerra, Juan de Lara, Francisco Jurado
Universidad Autónoma de Madrid (Spain)

***Abstract*—We are witnessing a rising role of mobile computing and social networks to perform all sorts of tasks. This way, social networks like Twitter or Telegram are used for leisure, and they frequently serve as a discussion media for work-related activities.**

In this paper, we propose taking advantage of social networks to enable the collaborative creation of models by groups of users. The process is assisted by *modelling bots* that orchestrate the collaboration and interpret the users' inputs (in natural language) to incrementally build a (meta-)model. The advantages of this modelling approach include ubiquity of use, automation, assistance, natural user interaction, traceability of design decisions, possibility to incorporate coordination protocols, and seamless integration with the user's normal daily usage of social networks.

We present a prototype implementation called SOCIO, able to work over several social networks like Twitter and Telegram, and a preliminary evaluation showing promising results.

***Index Terms*—Collaborative modelling; meta-modelling; social networks; natural language processing**

I. INTRODUCTION

According to recent studies [1], 70% of American citizens are users of social networks. People exploit social networks like Twitter, Telegram or Facebook, often on a daily basis, to be in contact with friends, share media, organize leisure activities, or discuss work-related issues in an agile manner.

The use of social networks for Software Engineering has been recognised as having high potential impact in practices and tools [2], [3]. One of the reasons is that they enable agile and lightweight means for coordination and information sharing. However, some identified open challenges include their use to increase community and end-user involvement, enhance project coordination, and improve development activities [3].

In this work, our goal is to profit from the benefits and potential of social networks to assist in a particular Software Engineering task: modelling and meta-modelling. For this purpose, we propose the collaborative modelling through social networks assisted by *modelling bots* that interpret the messages of users in order to construct a model or meta-model. User messages can be expressed either in natural language (NL), or they can be commands for model evolution. In the first case, the bot interprets the messages using NL processing techniques [4], [5] and derives update actions over the current model version. The approach is inherently collaborative, and integrates seamlessly with the normal usage of social networks to which users are familiar with. Moreover, it naturally leads to an accurate documentation of the traceability and provenance of the different design decisions incorporated into the model.

This paper motivates and presents usage scenarios for modelling bots, and presents a working prototype called SOCIO that is able to orchestrate model construction across both Twitter and Telegram. We have performed an initial user evaluation with promising results that encourage pursuing further research in this direction.

The rest of this paper is organized as follows. Section II motivates our approach, identifies envisioned scenarios, and elicits a set of requirements. Section III details the components of our proposal. Section IV describes architecture and tool support. Section V shows the results of our preliminary evaluation. Section VI compares with related work, and finally, Section VII ends with the conclusions and future work.

II. MOTIVATION AND USAGE SCENARIOS

The main motivation for this work is being able to benefit from the collaborative and ubiquitous nature of social networks – applications that people use on a daily basis – to perform assisted lightweight modelling. To this aim, we propose repurposing social networks based on micro-blogging (like Twitter or Telegram) as front-ends for the modelling activity, where dedicated bots interpret certain user messages to assist in the model construction. This way, the assisted modelling process seamlessly integrates with the normal use of social networks for discussion.

This approach enhances flexibility in modelling because it can be used in mobility and does not require installing new applications, but users can rely on apps they are already familiar with. When working on mobile devices, interacting via short messages can be easier and faster than using a diagramming tool, and can serve to quickly prototype models. Moreover, people with little or no background in computer science or modelling may be able to actively participate in modelling sessions. This may foster the collaboration of domain experts with teams of engineers. By recording the messages processed by the bot, the approach can trace information of the design decisions (*who* made *what*), so that every decision can be justified or rolled back.

This approach can be useful in several scenarios. First, to allow engineers quickly prototyping models *when* and *where* needed (e.g., in working meetings, but also when travelling home). Second, to assist teams of engineers collaborating with domain experts (who may lack a computer science background) to create domain models or meta-models. Third, in the educational domain, to enable groups of students the collaborative resolution of modelling exercises. In this scenario, bots could be configured for gamification activities or blended learning. Finally, being based on social networks,

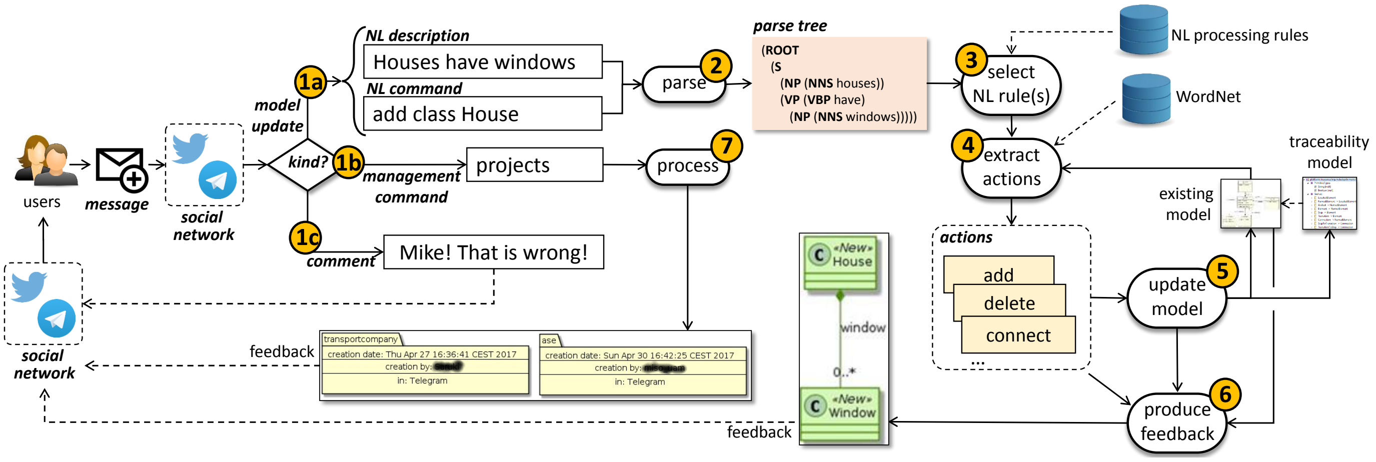


Fig. 1. Overview of our approach to assisted modelling via social networks

the modelling process can involve a large number of people. Hence, we foresee its use to crowdsource modelling decisions.

In order to give support to these scenarios, we identify the following requirements for our envisioned approach:

- Interaction through NL (to permit use by domain experts) and commands (more suitable for modelling experts). Anyhow, commands should have a flexible and natural syntax to minimize mistakes and user frustration.
- Traceability mechanisms able to justify design decisions and find their provenance.
- Integration of multiple social networks, so that users can use their preferred one.
- Support for both modelling and meta-modelling.
- Customizable collaboration protocols, e.g., supporting user roles and voting.
- Interoperability with accepted modelling frameworks, like the Eclipse Modelling Framework (EMF) [6].

Next, we detail our first steps towards realizing this vision.

III. Approach

Fig. 1 sketches our approach to modelling via social networks. Users can interact within the network of choice by sending messages directed either to the other partners (label 1c) or to the modelling bot (labels 1a-1b). The former messages permit discussing and coordinating, and they are handled by the network normally (i.e., they are regular text messages). Instead, messages directed to the bot are used for building models, and their format may depend on the particular social network. For instance, to send a message to the bot in Twitter, the message must mention the username of the bot (@modellingBot), while in Telegram, the message should start by /. The way of organising the collaboration also depends on the particularities of the social network. In Twitter, users of our system will normally be followers of the bot supervising the modelling process, so that every message sent to the bot will be received by its followers. In Telegram, users would create a group with the users interested in the modelling task, including the bot. In this way, every sent message will be received by all members of the group.

Messages directed to the bot are received by the users of the social network, just like any other message, but in addition, they are processed by the bot. We distinguish two kinds of such messages: *management commands* (label 1b) and *model update messages* (label 1a). The former allow performing project management tasks, like querying the existing modelling projects, creating a new model, or recovering the history of changes performed in a model. The bot processes such messages (label 7) and sends the result to the social network as a diagram embedded in an image file. The figure shows an example, where the set of available projects is returned as a package diagram (output of activity labelled 7). On their side, model update messages (label 1a) can be either *commands* or *descriptions*. The former are imperative actions to directly manipulate a model, e.g., to add a class or feature, change its type, or delete an element. The latter are descriptive statements of the domain concepts, like “houses have windows”.

Both commands and descriptions are expressed in NL. Hence, they are processed by a NL parser (label 2) which produces a parse tree of the sentences in the message. Supporting commands in NL provides flexibility because there is no need to adhere to a strict syntax. The system has an extensible library of NL processing rules, able to handle different kinds of NL phrases (label 3); currently, we support rules targeted to meta-model construction. From the current model state and the information inferred from the message, we synthesize a number of model update actions (label 4). As our approach is incremental, sometimes it is necessary to refer to existing model elements. In order to provide flexibility and avoid redundancies, we allow for synonyms which are sought using WordNet [7], a lexical database for the English language. We will describe our NL processing approach in Section III-A, and the extraction of model update actions in Section III-B.

The extracted actions are applied to the current version of the model to make it evolve. Moreover, the system maintains a traceability model to keep track of *why* the model was updated, and by *who* (label 5). We will explain model building and traceability in Section III-C. Finally, the bot emits a feedback message (label 6), which is received through the social network. The feedback to model update messages consists of an image of the updated model, with annotations indicating the last modifications. Section III-D will illustrate some steps in a model construction session, and the feedback obtained.

A. Natural language processing

We use the Stanford NL parser [5] to process model update messages, both commands and descriptions. The parser creates a parse tree with the grammatical relations of the words in the message, as Fig. 1 shows for message "houses have windows" (see output of activity labelled 2). This tree identifies the noun phrases (NP) that may provide information about the model, and the syntactical role of each part of the phrase. For example, in the tree of Fig. 1, "houses" and "windows" are tagged as common plural nouns (NNS), while "have" is tagged as a verb in present tense which is non-3rd person singular (VBP).

The parsed messages are interpreted according to a number of rules, which we currently base on the work of Arora and collaborators [4]. Each rule specifies the combination of word classes that activate the rule, usually based on the presence of certain verbs, as well as the model update actions that should be performed when the rule is matched. Our rules currently handle the construction of meta-models only, but since our approach is extensible, we plan to expand the set of rules to tackle the construction of models, likely instances of previously defined meta-models using this same approach. We consider the following rules:

Verb to be: When the main verb of the phrase is "to be", it can indicate either an inheritance relation between two classes (e.g., "Kitchen is a room", "Service may be premium service or normal service"), or the type of a feature (e.g., "Name is string", "The bank of the customer is BLUX"). If the phrase contains an expression of the form "A of B" or a genitive "B's A", then the rule infers that A is an attribute or reference of B.

Verb to have: When the main verb is "to have", or synonyms of it like "characterized by" and "identified by", this rule infers the subject of the sentence is a class, which has a feature. Deciding whether the feature is a reference or an attribute depends on the information there is in the meta-model about the feature. If there is not enough information, the feature is assigned an "open" type which may be refined by subsequent messages. As an example, the message "Bulky packages are characterized by their width, length and height" triggers the creation of features width, length and height with open type in class BulkyPackage.

Transitive verb: This rule handles all verbs with a subject and a direct object. It creates classes for the subject and direct object, and a reference whose name is the verb. For example, the phrase "The simulator shall send log messages" triggers the creation of classes Simulator and LogMessage, and the reference send from the former to the latter.

Contain: Verbs like "contain", "be made of", "include" and "be composed of" imply a composition relation between two classes. For example, the phrase "A delivery is made of packages" creates a composition relation between the classes Delivery and Package, and also creates the classes if they do not exist yet.

Add: This rule handles imperative sentences (with implicit subject) whose verb is "add", "create", "make", etc. These are interpreted as commands with a flexible syntax, resulting in the creation of classes, attributes or references. For example, "add house" will create the class House, while "create room in house" adds a feature room to class House.

Remove: This rule is similar to the previous one but for deletion. It considers imperative verbs synonyms of "remove", like "delete" and "erase".

Model update messages can include several sentences and more than one verb, like in "Add house and remove windows". Moreover, the processing of one message may trigger several NL rules, in which case, we apply the rule with higher priority. In particular, rules seeking for specific verbs have higher priority than the more general rule seeking transitive verbs.

B. Model update actions

As abovementioned, each NL rule specifies the model update actions to be applied when the rule is selected. The possible actions are the following:

Add class: This action is issued when the rule finds a common name that should be a class. The class is not created if one exists with the same or synonym name. Supporting synonyms provides flexibility and avoids redundancy. We apply accepted modelling styles for class names (i.e., singular, camel case).

Make class abstract/concrete: Classes can be set to abstract or concrete using their name or a synonym. If the class does not exist, then an *add class* action is issued as well.

Set parent class: This action sets an inheritance relation between two classes, creating the classes if they do not exist.

Remove parent: This action removes an inheritance relation. If the class does not exist, the action will make no changes.

Add attribute: This action is issued by the "verb to have" rule (e.g., "packages have weight") and in case of genitive cases (e.g., "package's name"). The attribute is added to the given class or to a synonymous one, creating a new class if it does not exist. If the class already owns a reference with same name, it is replaced by an attribute. The attribute's upper cardinality is set to 1 if the attribute name is singular, or to * if it is plural. At this point, the attribute type is left open, so that it can be refined later.

Add reference: This action is issued by the "transitive verb" and "contain" rules, and it works similarly to the previous action. If the owner class already defines an attribute with the same name, it is replaced by a reference.

Modify feature type: We support primitive data types like int, float, String, boolean and Date for attributes, while the type of references must be a class. The feature is created if it does not exist.

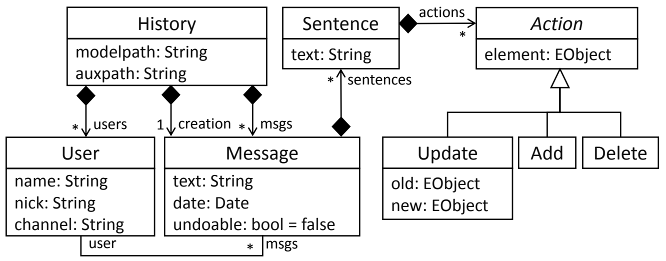


Fig. 2. Traceability meta-model

Remove class: It removes a class and its features.
Remove feature: It removes one or several features.

Any action can be undone and redone through commands.

C. Model update and traceability

The actions derived from the messages are applied to the current model version. In some cases, these actions may lack some information. For instance, the action that adds an attribute may miss the specific type of the attribute, in which case, its type is left "open" so that it can be refined later. Similarly, as removing a class would let the references pointing to the class dangling, we add a provisional "ghost" class as target of these references. We foresee an automated mechanism that completes all these "open" design decisions with sensible defaults in a final stage.

We also maintain traceability information of each message sent to the bot, including the sender and the model update actions it triggers. We use a model-based approach to record the traceability data, building a traceability model conformant to the meta-model in Fig. 2 for each model being constructed. Class User keeps track of the participant users and the social network (channel) they employed to send the messages. We store all messages directed to the bot, and distinguish the message used to create the model. Actions point to the model elements affected by the action using reference element, whose type is EObject as this is the base class in EMF, the implementation technology we use. To keep track of the removed elements which are no longer in the model, we store them in an auxiliary model. Finally, Update actions point to the old and new versions of the updated element in the auxiliary model, and to the current version of the element in the model.

An image of the updated model diagram is sent to the collaborator users. The modified model elements are marked in the diagram, and a note explains the performed changes. This information is read from the traceability model. The trace can also be queried using the management command history, which sends a diagram with the traceability information to the users.

D. Example

Fig. 3 illustrates a typical modelling session. The rectangles labelled 1–4 contain NL messages that a user sends, while the diagrams are the feedback provided by the modelling bot.

The first message is handled by the "transitive verb" rule. This creates classes for "good transport company" and "delivery", and a reference for "handle". The cardinality of the

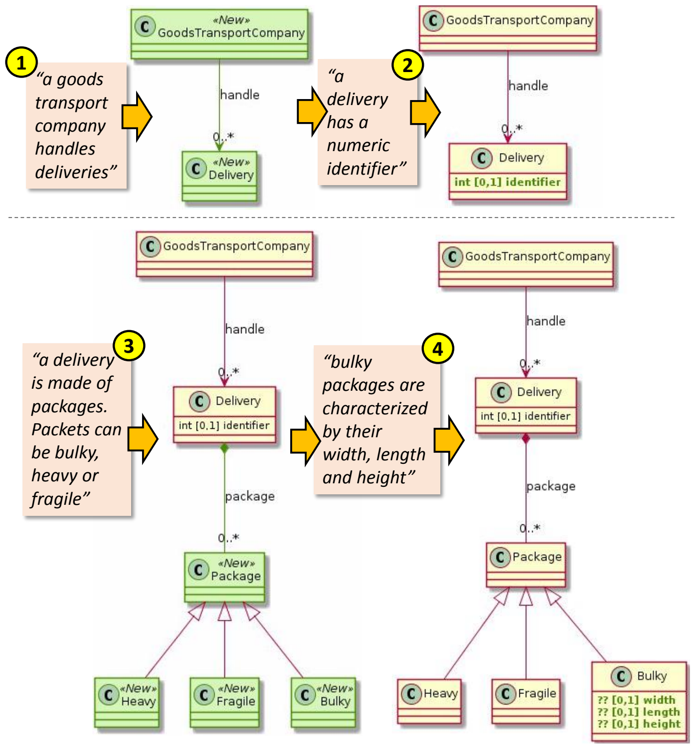


Fig. 3. Some steps in a model construction session

reference is many as "deliveries" is in plural. The created classes have singular, camel case names. The newly created elements are shown in green. For space constraints, the figure omits the explanation of changes which the bot also produces as notes in the diagram.

The second message is handled by the "verb to have" rule, which adds an integer attribute to class Delivery. The bot assigns an upper cardinality of 1, as there is no plural.

The third message contains two phrases. The first one is handled by the "contain" rule, which creates class Package and reference package. The second one is handled by the "verb to be" rule, which creates the inheritance hierarchy. This phrase makes use of the word "packet", which is as a synonym of "package", and hence, no new class is added for it.

The fourth message, processed by the "verb to have" rule, creates three features in class Bulky. At this point, there is no information on whether they should be attributes or references, and hence, this is left open (shown with "??").

IV. Tool Support

We have developed a prototype tool for our approach called SOCIO (from assisted modelling through social networks). Fig. 4 shows its architecture. The tool supports Twitter and Telegram, though it can be extended with further social networks by implementing an interface. This means that users of different social networks can interact with each other.

Independently of their provenance, all bot-processable messages are enqueued and processed one-by-one. NL processing is performed using the Stanford parser and WordNet. The cre-

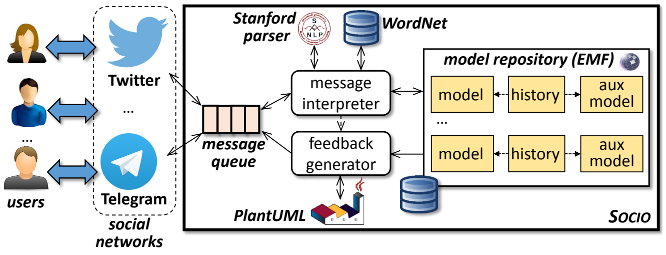
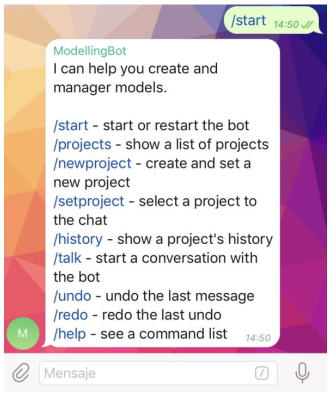


Fig. 4. Architecture of our tool Socio



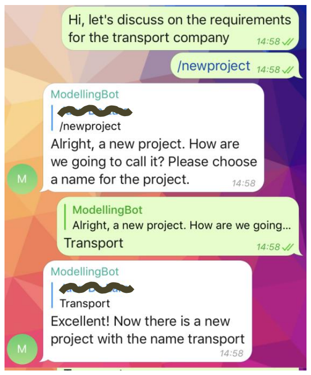
/start 14:50

ModellingBot
I can help you create and manager models.

/start - start or restart the bot
/projects - show a list of projects
/newproject - create and set a new project
/setproject - select a project to the chat
/history - show a project's history
/talk - start a conversation with the bot
/undo - undo the last message
/redo - redo the last undo
/help - see a command list 14:50

Mensaje

(a) initializing the bot (Telegram)



Hi, let's discuss on the requirements for the transport company 14:58

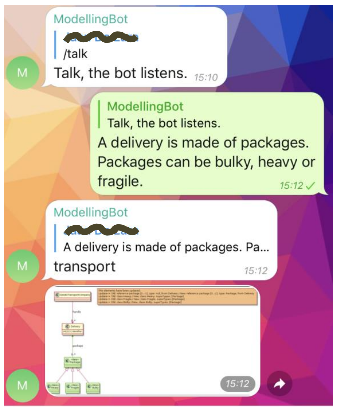
/newproject 14:58

ModellingBot
/newproject
Alright, a new project. How are we going to call it? Please choose a name for the project. 14:58

ModellingBot
Alright, a new project. How are we going...
Transport 14:58

ModellingBot
Transport
Excellent! Now there is a new project with the name transport 14:58

(b) discussion and project creation



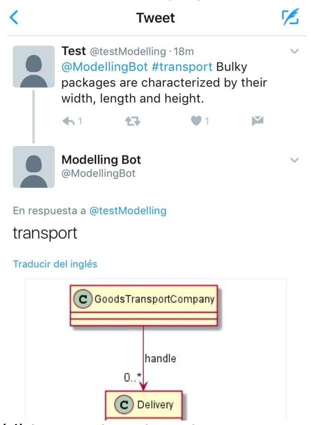
ModellingBot
/talk
Talk, the bot listens. 15:10

ModellingBot
Talk, the bot listens.
A delivery is made of packages. Packages can be bulky, heavy or fragile. 15:12

ModellingBot
A delivery is made of packages. Pa...
transport 15:12

15:12

(c) providing NL descriptions



Tweet

Test @testModelling · 18m
@ModellingBot #transport Bulky packages are characterized by their width, length and height.

Modelling Bot
@ModellingBot

En respuesta a @testModelling
transport

Traducir del inglés

(d) interaction via Twitter

Fig. 5. Some steps in a modelling session via Telegram (a-c) and Twitter (d)

ated models are stored using EMF [6], the de-facto modelling standard nowadays. Socio currently supports the collaborative construction of meta-models, but we plan to give support to the creation of instance models in the future. Due to the message length limit of widespread social networks, the feedback is frequently given as a diagram image that may include the current model state, its history, or the set of available projects. These images are generated using PlantUML [8].

To illustrate the tool's capabilities, Fig. 5 shows a sample modelling session over Telegram and Twitter. It reflects the discussion of a set of engineers and the bot for building a meta-model for a transport delivery company.

Steps (a-c) correspond to the interaction in a Telegram group to which the engineers and the bot belong. The bot is started in step (a), and it replies with all available commands. This is a normal message that is received by all members of the group. Step (b) shows one discussion message between the engineers, which is interchanged using the social network normally. It also shows a message directed to the bot, asking the creation of a new model project. In step (c), a NL message describing requirements for the meta-model is directed to the bot. This is performed using the \talk command, after which the bot prompts the user to talk, and the user replies with a NL sentence. The bot processes the sentence, updates the meta-model, and returns an image of it. The image can be enlarged and shared with other Telegram groups and users via email or external services like Dropbox. Step (d) shows the interaction with the bot using Twitter. This requires mentioning the bot account and the model project. Interestingly, in addition to the traceability provided by Socio using the meta-model of Fig. 2, the organization of messages in a life-line which can be browsed at convenience, as provided by Telegram, is also an excellent means to track down design decisions.

V. Evaluation

To assess the suitability of our proposal, we have conducted a preliminary evaluation with 10 participants organized in 4 Telegram groups: 2 groups of 2 people, and 2 groups of 3 people. They were asked to create a meta-model for e-commerce in 15 minutes but with no other restriction, and then complete a questionnaire with 3 parts: two with Likert-scale questions, and a last one with free text questions.

All participants had a computer science background (post-graduate or last year degree students) and were non-native English speakers. The average declared modelling expertise was 62,5% (out of a maximum of 100%), and the level of English was 72,5%. Six conducted the task using the mobile, 2 the web browser, and 2 the desktop Telegram application.

The first part of the questionnaire consisted of the 10 questions of the System Usability Scale (SUS) [9], a de-facto standard to measure system usability. Socio obtained 74%, which indicates good usability. Interestingly, the users that gave the lowest SUS scores were those with less modelling expertise or level of English (the bot must be addressed in English). In particular, the Pearson correlation coefficient was 0,660 with a significance level 0,038.

The second part of the questionnaire comprised 8 questions evaluating four aspects: (1) suitability of NL to build models w.r.t. using an editor, (2) precision of the bot to interpret NL, (3) enough functionality in the command set, and (4) whether they liked embedding a modelling tool in a social network, or they would prefer a separate collaborative tool. We obtained around 75% for aspects 1 and 4, indicating that participants considered NL as a suitable interaction mechanism, and they appreciated the idea of collaborating through social networks. Aspect 3 was rated 60%, which is acceptable but leaves room for enriching the command set. Regarding aspect 2, the statement "bot-generated models agree with the provided NL phrases" was scored 62,5%. However, all participants observed some mismatch between the NL phrases and the obtained models, which suggests the need to improve the precision of the bot to interpret NL. Again, the users assigning a low score were those with less modelling or English expertise.

In the free-text questions, several participants identified as positive the possibility to use the tool on the phone, and being fun, easy-to-use and quicker than other modelling tools. They also suggested some improvements, like the need for coordination mechanisms, and commands to change the reference cardinalities. We will tackle this in future work.

Regarding interaction, participants used more often descriptive NL to interact with the bot (80%) than imperative, command-like messages (20%). This suggests that NL was found useful to complete the task. Overall, 50% were discussion messages and 50% were bot-directed messages. Talking to the bot was balanced in all groups, but in one group where a participant took the role of coordinator and basically sent messages to the other participants. This need for discussion justifies the inclusion of the modelling tool in a social network.

The study is preliminary, with several threats, like the low number of participants, the limited group size, the similar participant background, the fact that participants were non-native speakers, and the lack of a precise modelling goal which permitted evaluating the produced artefacts. However, the positive results encourage further research on this approach.

VI. Related Work

The impact and potential of collaborative and participatory modelling has been recognised by several disciplines – like water resources management and sustainable development [10], or smart product design [11] – to enhance their modelling and decision-making processes. However, typically these works do not propose a concrete method or tool.

In the context of Software Engineering, collaborative modelling has been used for model construction [12] and collaborative creation of domain-specific languages [13]. However, these approaches do not use social networks or NL processing. Instead, they rely on collaborative graphical model editors [12] or ad-hoc tools [13], with no assistant support.

Nowadays, people are used to social networks, and this fact has made organizations to adapt and introduce a social network perspective within their development processes [14], [15]. In this sense, an interesting example of a distributed problem-solving model that combines human and machine computation is crowdsourced software engineering [16]. Many commercial platforms like TopCoder (www.topcoder.com), Bountify (bountify.co) or uTest (www.utest.com) permit recruiting online labour to work on specific tasks, like coding and testing. However, as far as we know, there is no proposal on assisted collaborative modelling over social networks.

Our work proposes using NL to both human collaboration and bot interaction. NL processing techniques have been used within Software Engineering to derive UML diagrams/domain models from text [4], [17]. Our contribution in this context is to use an interactive, incremental approach, and the use of social networks to embed both assistance and collaboration.

Altogether, the use of social networks for collaborative modelling based on NL processing is a novel research direction.

VII. Conclusions and Future Work

In this paper, we have proposed a novel approach to collaborative modelling via social networks, with assistant bots able to process NL messages. We have created prototype tool support working over Telegram and Twitter, and performed an initial evaluation obtaining encouraging results.

In the future, we will incorporate customizable collaboration protocols for different styles of decision making, e.g., based on votings, or roles. We will develop support for building instances of meta-models, and for querying the model elements provenance. We will consider other types of bots, e.g., a quality assurance bot which monitors the current model state to suggest improvements, or gamification bots. We plan to use WordNet's super-subordinate relations to propose inheritance relations, as well as investigate the use of speech recognition for modelling. Finally, we foresee developing scalability mechanisms, e.g., by pruning the bot feedback to return only the modified model elements and their context.

Acknowledgment

Work funded by the Spanish MINECO (TIN2014-52129-R) and the R&D programme of Madrid (S2013/ICE-3006).

References

[1] Pew Research Center, `http://www.pewinternet.org/fact-sheet/social-media/`.

[2] A. Begel, R. DeLine, and T. Zimmermann, "Social media for software engineering," in *Proc. FoSER@FSE*. ACM, 2010, pp. 33–38.

[3] M. D. Storey, C. Treude, A. van Deursen, and L. Cheng, "The impact of social media on software engineering practices and tools," in *Proc. FoSER@FSE*. ACM, 2010, pp. 359–364.

[4] C. Arora, M. Sabetzadeh, L. C. Briand, and F. Zimmer, "Extracting domain models from natural-language requirements: approach and industrial evaluation," in *Proc. MoDELS*. ACM, 2016, pp. 250–260.

[5] M. Marneffe, B. Maccartney, and C. Manning, "Generating typed dependency parses from phrase structure parses," in *Proc. LREC*, 2006.

[6] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework 2.0*, 2nd ed. Addison-Wesley Professional, 2009.

[7] G. A. Miller, "Wordnet: A lexical database for english," *Comm. ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[8] Plant UML, `http://plantuml.com/`.

[9] J. Brooke, "SUS: A retrospective," *J. Usability Studies*, vol. 8, no. 2, pp. 29–40, 2013.

[10] M. Hare, "Forms of participatory modelling and its potential for widespread adoption in the water sector," *Environmental Policy and Governance*, vol. 21, no. 6, 2011.

[11] T. Ahram, W. Karwowski, and B. Amaba, "Collaborative systems engineering and social-networking approach to design and modelling of smarter products," *Behav. Inf. Tech.*, vol. 30, no. 1, pp. 13–26, 2011.

[12] J. Gallardo, C. Bravo, and M. A. Redondo, "A model-driven development method for collaborative modeling tools," *J. Network and Computer Applications*, vol. 35, no. 3, pp. 1086–1105, 2012.

[13] J. L. C. Izquierdo and J. Cabot, "Collaboro: a collaborative (meta) modeling tool," *PeerJ Computer Science*, vol. 2, p. e84, 2016.

[14] F. O. Zaffar and A. Ghazawneh, "Knowledge sharing and collaboration through social media - the case of IBM," in *Proc. MCIS*, 2012.

[15] C. Manteli, H. Vliet, and B. Hooff, "Adopting a social network perspective in global software development," in *Proc. ICGSE*, 2012, pp. 124–133.

[16] T. D. LaToza and A. van der Hoek, "Crowdsourcing in software engineering: Models, motivations, and challenges," *IEEE Software*, vol. 33, no. 1, pp. 74–80, 2016.

[17] M. Landhäußer, S. J. Körner, and W. F. Tichy, "From requirements to UML models and back: How automatic processing of text can support requirements engineering," *Software Quality Journal*, vol. 22, no. 1, pp. 121–149, 2014.