Modelling Mutually Interactive Fictional Character Conversational Agents

Thomas Winters[0000−0001−7494−2453]

KU Leuven, Leuven, Belgium,
thomas.winters@cs.kuleuven.be

Abstract. Conversational agents, such as chatbots and virtual assistants, are typically modelled to have a broad, generic personality, which they employ in their communication with single human beings. However, by framing a conversational agent as existing fictional characters, humans can imagine a shallow agent to have a larger personality than without this framing. Using multiple such agents allows for conversational interactions that help construct stories with or without human intervention, leading to multi-agent human-computer interactive story telling. In this paper, we model six semi-independent Twitterbots based on fictional characters based on the Belgian children's TV show Samson & Gert, which are mutually interactive with each other as well as with other Twitter users. To achieve this, we first introduce a new language for modelling generative weighted context-free grammars called Babbly and a new framework for easily specifying complex Twitterbot behaviour. We found that these bots were not only well received by users, but also created lots of interesting, unexpected positive interactions. Using fictional characters as framing for conversational agents can thus help achieving interesting personalities and shows potential in interactive computational story telling.

Keywords: Chatbot · Twitterbot · Generative Models · Computational creativity · Computational humor

1 Introduction

Conversational agents have seen a large increase in commercial success, with most big tech companies developing their own virtual assistant, such as Alexa, Google Assistant, Siri and Cortona. All these assistants have chosen to model generic personalities, due to having to fulfill a wide variety of tasks. This large variety of tasks implies that these agents are usually used as a tool, and by extend only in conversations between two entities, namely the conversational agent and a human [2].

In this paper, we propose one way of adding more personality by adding personality traits of fictional characters, and making them interact with other bots based on fictional characters from the same world. This framing can help humans to induce more characteristics of the agent they are talking to, thanks to knowing their background. This might help creating the feeling of a group of friends, which might contribute to perceiving the conversational agent(s) as more human. We use the Belgian children's TV show *Samson & Gert* as a source of fictional characters, as they have a wide range of figures with varying signature lines.

2 Background

2.1 Conversational Agents

Conversational agents have been researched for several decades, with many frameworks offering different approaches for having conversations. The Turing test [13] was proposed in 1950 as a way of testing the thinking capabilities of machines by making humans judge if they were conversing with another human or with a machine. This influential (but also controversial) test was later implemented in the yearly Loebner prize competition, which is won by the conversational agent that fools the judges the most. One of the most frequent and famous early winners in the Loebner prize is ALICE, and more recently Mitsuku, which is in a way an extension of ALICE [1,15]. Both chatbots aim to achieve realistic dialogue using mark-up language AIML (Artificial Mark-up language), a derivative of XML. This open-source language allows botmakers to create dialogue patterns with wildcards, which match input to output dialogue by finding the best fitting input pattern [12]. Although this is a very time consuming task to model, developers can thus use AIML to produce a chatbot that responds with sensible answers using correct grammar. The quality of this type of chatbot typically improves by limiting the domain, e.g. on a frequently asked questions (FAQ) catalogue, to train the bot [12].

Twitter is relatively new medium for chatbots to flourish, although these bots are usually called *"Twitterbots"* on this platform [14]. One of the most popular frameworks for designing Twitterbots is Tracery, a language for specifying generative context-free grammars [4], using CBDQ[1]. Although many thousands of Twitterbots have been created using this framework, they are usually unable to adapt to user replies, and just tweet like they usually would in response to user input.

2.2 Framing in Computational Creativity

Framing is one of the four aspects (along with Aesthetics, Concept and Expression) of the influential computational creativity framework for categorising and evaluating creativity in machines, called FACE [3]. It denotes artefacts that help understand generated artefacts by a system, or as it has been recently redefined: *"Framing refers to anything (co-)created by software with the purpose of altering*

[1] https://cheapbotsdonequick.com/

an audience's perception of a creative work, or its creator." [6]. It is an underused aspect in generative systems, but has been shown to enhance the perceived creativity in systems [6]. We believe that by framing our conversational bots as fictional characters, and making them interact with each other as generated framing, improve their perceived capabilities.

3 Framework Implementation

3.1 Babbly: Language for Modelling Generative WCFGs

Babbly[2] is a new language we introduce for easily modelling generative weighted context-free grammars. The syntax is similar to what one would expect from a language for modelling generative context-free grammars, but with the addition of allowing clauses to be weighted, and allowing several regular expression-like constructs and records. For example, in Figure 1, we can see several of these allowed language constructs.

```
person = {
    4: Gertje,
    1: Bobientje
}
ao = (a|o)
mwahzeg = Mw<ao:>{1,3}h (zeg hé(, <person>){.6}|<person>)
```

Fig. 1. Basic Babbly program for generating Samson's catchphrase

The Babbly code in Figure 1 declares that the named generator `person` has a probability of 80% of generating *"Gertje"* and otherwise *"Bobientje"*. The named generator[3] `ao` will uniformly pick *"a"* or *"o"* when used. These named generators are referenced using triangular brackets, and can be locked to always return the same value using a record. A record is denoted by using a colon after the name of the declaration, followed by a record name of arbitrary length (even zero, as used in `<ao:>`). Curly brackets are used to denote the repetition of a part, similar to their use in regular expressions syntax. As such, `<ao:>{1,3}` is equivalent to `<ao:>|<ao:><ao:>|<ao:><ao:><ao:>`, and will thus generate *"a"*, *"aa"*, *"aaa"*, *"o"*, *"oo"*, *"ooo"*. Note that without the colon, which binds the construct to a record, *"aoa"* would also have been a possible generation. When only a single fraction between 0 and 1 is used between curly brackets, this will return the part between brackets with this probability, and the empty string otherwise. As

[2] https://github.com/twinters/babbly
[3] We prefer to use the term *"named generator"* over *"variable"* since the assigned value does not change over time.

such, the total probability the code in Figure 1 generating for example *"Mwaah zeg hé, Gertje"* is $\frac{1}{6} \cdot \frac{1}{2} \cdot \frac{6}{10} \cdot \frac{4}{5} = \frac{1}{25}$, and the probability of *"Mwoh Bobientje"* is $\frac{1}{6} \cdot \frac{1}{2} \cdot \frac{1}{5} = \frac{1}{60}$.

More advanced constructs that the language offers is the support of externally defined named generators, generator cascades and (possibly user-defined) functions, as can be seen in Figure 2. In this figure, both `lastname` and `firstname` are external named generators given to Babbly upon running the generator. These are transformed by the externally defined function `mispronounce`, and then capitalised by the Babbly functions `capitaliseAll`. The slash in between denotes that the text generator will first check if a last name is given, and if not, just use the `<firstname.mispronounce>` to generate `person`.

```
person = Meneer <lastname.mispronounce.capitaliseAll>/<firstname.mispronounce>
```

Fig. 2. Example of some advanced Babbly features, which redefines the `person` declaration of Figure 1.

Comparison with Tracery Although Tracery is an efficient and intuitive framework that makes building text generators more accessible to a wide audience irregardless of their background in computer science, our novel text generation language Babbly provides several significant advantages over other popular text generation frameworks like this. First, since it allows for adding weights to clauses, it is much easier to model low probability possible generations, as Tracery users usually have to duplicate clauses multiple times to increase the weight, leading to more overhead when changing these clauses later. Secondly, Babbly offers records for locking certain instances of a named generator, such that a generation can more easily be consistent with its references without the need of declaring functions. Thirdly, Babbly offers support for importing other Babbly files as well as newline separated word lists as variables, meaning that named generators and word lists can easily be shared between different text generators. Fourthly, it offers shorthands inspired by regular expressions, e.g. disjunction, repeats and probability of occurring, to remove the need of introducing unnecessary variables. Fiftly, it supports loading in externally specified texts as named generators, and treating it as any other Babbly-defined text generator. This way, our system allows to insert words that are relevant in the conversation a bot might be having with a user. Sixtly, the syntax of the language is designed to be expressive enough to quickly model many types of text generators, instead of using JSON syntax like Tracery does. One downside of this is that JSON-based text generators are usually easier to port to other platforms and have good support in many editors due to their limited syntax rules. We implemented a syntax highlighter in Sublime Text to decrease the possibilities of mistakes when modelling text generators in Babbly.

Comparison with AIML Compared to chatbot frameworks like AIML, Babbly is more focused on generating a diverse set of versions of a particular answer, while AIML has the functionality to know which replies work for what kind of response. This is due to Babbly missing structures like AIML's patterns to match input. There is also no parsing of the input and remembering topics mentioned in the conversation. This means that these tasks have to be done by external systems refering to the appropriate reply templates, e.g. the behaviour objects explained in section 3.2. However, AIML only offers a uniform random selection of possible answers to a prompt, whereas Babbly offers more flexible templates encoding many possible answers, with tighter control over the probability of every possible answer. Babbly also offers external functions to help determine certain aspects of the generated texts. It is thus more focused on the generative side than the reply side, and could thus be used in a more pattern-based reply system to determine the appropriate template to respond.

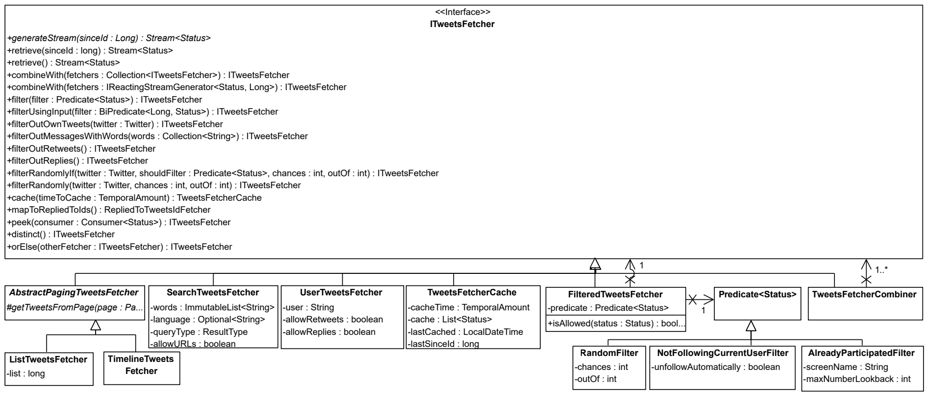
3.2 Twitter Agent Behaviour Modelling Framework

We introduce a new framework[4] that serves as a layer on top of the popular Twitter4J[5] library by creating abstractions for simplifying the modelling of complex Twitterbot behaviours.

TweetFetchers We introduce an interface called `TweetsFetcher` that generates streams of tweets upon calling with an optional parameter of tweets being more recent than a particular tweet identifier. An instance of a `TweetsFetcher` has already encoded the necessary variables to know what types of tweets should be fetched from Twitter (e.g. a search query). There are many types of `TweetsFetchers` available that are built on Twitter4J's functionalities, such as fetching mentions, timeline, tweets corresponding to a search, hashtag, user or of a particular twitter list. There are also fetchers that use other `TweetsFetchers`, e.g. by caching them, fetching the tweets of which tweets from another stream reply to, combining streams together and for cascading fetchers (meaning that the second `TweetsFetcher` is only called if the first one returns an empty stream of tweets). The last category of `TweetsFetchers` is for the fetcher filters, which filter out particular tweets from a stream, e.g. filter out particular tweets based on a given condition, at random, already replied tweets or tweets from users that are not following the bot. This is thus a powerful abstraction to model the constraints on the streams of tweets a particular Twitterbot behaviour can reply to.

TwitterBehaviours Our framework introduces the notion of `PostBehaviour`, `ReplyBehaviour` to model behaviours and compose composite Twitterbot behaviours. Upon execution of a Twitterbot in this framework, the executor calls the `PostBehaviour` a given number of times during a given timerange at random

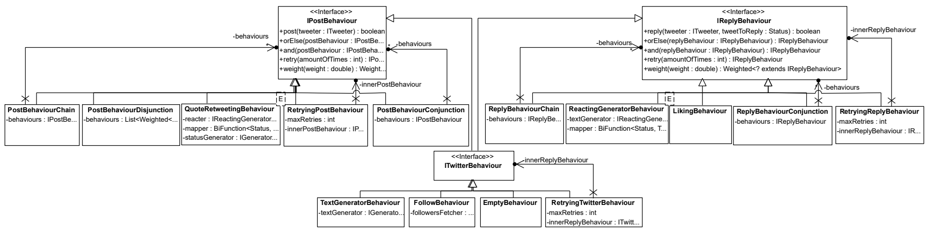
[4] https://github.com/twinters/twitter-util
[5] http://twitter4j.org



<<Interface>>
ITweetsFetcher
+generateStream(sinceId : Long) : Stream<Status>
+retrieve(sinceId : long) : Stream<Status>
+retrieve() : Stream<Status>
+combineWith(fetchers : Collection<ITweetsFetcher>) : ITweetsFetcher
+combineWith(fetchers : IReactingStreamGenerator<Status, Long>) : ITweetsFetcher
+filter(filter : Predicate<Status>) : ITweetsFetcher
+filterUsingInput(filter : BiPredicate<Long, Status>) : ITweetsFetcher
+filterOutOwnTweets(twitter : Twitter) : ITweetsFetcher
+filterOutMessagesWithWords(words : Collection<String>) : ITweetsFetcher
+filterOutRetweets() : ITweetsFetcher
+filterOutReplies() : ITweetsFetcher
+filterRandomlyIf(twitter : Twitter, shouldFilter : Predicate<Status>, chances : int, outOf : int) : ITweetsFetcher
+filterRandomly(twitter : Twitter, chances : int, outOf : int) : ITweetsFetcher
+cache(timeToCache : TemporalAmount) : TweetsFetcherCache
+mapToRepliedToIds() : RepliedToTweetsIdFetcher
+peek(consumer : Consumer<Status>) : ITweetsFetcher
+distinct() : ITweetsFetcher
+orElse(otherFetcher : ITweetsFetcher) : ITweetsFetcher
1
1..*
AbstractPagingTweetsFetcher
#getTweetsFromPage(page : Pa...
SearchTweetsFetcher
-words : ImmutableList<String>
-language : Optional<String>
-queryType : ResultType
-allowURLs : boolean
UserTweetsFetcher
-user : String
-allowRetweets : boolean
-allowReplies : boolean
TweetsFetcherCache
-cacheTime : TemporalAmount
-cache : List<Status>
-lastCached : LocalDateTime
-lastSinceId : long
FilteredTweetsFetcher
-predicate : Predicate<Status>
+isAllowed(status : Status) : bool...
1
Predicate<Status>
TweetsFetcherCombiner
ListTweetsFetcher
-list : long
TimelineTweets Fetcher
RandomFilter
-chances : int
-outOf : int
NotFollowingCurrentUserFilter
-unfollowAutomatically : boolean
AlreadyParticipatedFilter
-screenName : String
-maxNumberLookback : int

Fig. 3. Schematic overview of the main `TweetFetchers`

intervals. This `PostBehaviour` can be constructed using a text generator modelled using Babbly, a quote retweeter listening to a `TweetsFetcher`, a cascade or combination of other `PostBehaviours` or any other zero-argument function returning an optional string The `ReplyBehaviour` of the Twitterbot is called when finding a new unhandled tweet from the given `TweetsFetcher` that it listens to. This behaviour could also be a Babbly generator, an automatic follower, automatic liker, a combination or cascade of multiple `ReplyBehaviours` or any other any function mapping a tweet to an optional string.



<<Interface>>
IPostBehaviour
+post(tweeter : ITweeter) : boolean
+orElse(postBehaviour : IPostBe...
+and(postBehaviour : IPostBeha...
+retry(amountOfTimes : int) : IPo...
+weight(weight : double) : Weight...
-behaviours
-behaviours
-innerPostBehaviour
PostBehaviourChain
-behaviours : IPostBe...
PostBehaviourDisjunction
-behaviours : List<Weighted<...
QuoteRetweetingBehaviour
-reacter : IReactingGenerator...
-mapper : BiFunction<Status, ...
-statusGenerator : IGenerator...
E
RetryingPostBehaviour
-maxRetries : int
-innerPostBehaviour : IP...
PostBehaviourConjunction
-behaviours : IPostBehaviour
<<Interface>>
IReplyBehaviour
+reply(tweeter : ITweeter, tweetToReply : Status) : boolean
+orElse(replyBehaviour : IReplyBehaviour) : IReplyBehaviour
+and(replyBehaviour : IReplyBehaviour) : IReplyBehaviour
+retry(amountOfTimes : int) : IReplyBehaviour
+weight(weight : double) : Weighted<? extends IReplyBehaviour>
-behaviours
-innerReplyBehaviour
-behaviours
ReplyBehaviourChain
-behaviours : IReplyBe...
ReactingGeneratorBehaviour
-textGenerator : IReactingGene...
-mapper : BiFunction<Status, T...
E
LikingBehaviour
ReplyBehaviourConjunction
-behaviours : IReplyBehaviour
RetryingReplyBehaviour
-maxRetries : int
-innerReplyBehaviour : IR...
<<Interface>>
ITwitterBehaviour
-innerReplyBehaviour
TextGeneratorBehaviour
-textGenerator : IGenerato...
FollowBehaviour
-followersFetcher : ...
EmptyBehaviour
RetryingTwitterBehaviour
-maxRetries : int
-innerReplyBehaviour : ITwitt...

Fig. 4. Schematic overview of the `PostBehaviours` and `ReplyBehaviours`

4 Modelling Fictional Character Agents

Using our novel Babbly and Twitterbot framework, we implemented six different Twitterbots[6], each one based on a different character from the Belgian children TV show Samson & Gert show. Every Twitterbot is running seperately on a different server, with no shared real-time information, other than the Twitterfeeds

[6] List of the bots: https://twitter.com/thomas_wint/lists/samson-bots

they independently follow. The only exception to this is that the Twitterbots based on Samson and on Gert are running on the same server, as the GertBot needs to know the original word that SamsonBot mispronounced. All of these bots are interacting with each other, as well as with other Twitter users that interact with them.

4.1 SamsonBot

In the Samson & Gert TV show, Samson is a talking dog that is the main character of the show. His most prominent personality trait is that he does not know what most difficult words mean, and asks Gert what a word means while mispronouncing the word by saying similar sounding words. For example, if someone were to mention a *"Twitterbot"*, he might ask Gert what a *"Glitter pot"* is. Similarly, he mispronounces the names of almost all of the other characters in an analogous fashion.

Mispronunciation Function We achieved this mispronunciation behaviour in SamsonBot[7] by modelling a mispronunciation function and adding it as a custom function to the Babbly generator modelling Samson's speech patterns. This mispronunciation functionality minimises a weighted Levenshtein distance [8,9] of the original word to concatenations of at most two random words that do not form exactly the same word. The costs of the Levenshtein distance function are modelled to be higher for replacing or removing vowels than consonants, as the former contribute more to the general sound of the word and resemble the way Samson usually mispronounces words in the show. Additionally, it makes sure that the last letter of the word is the same, and if not, tries to fix this with as little change as possible.

Bot Behaviour The `TwitterBehaviours` of Samson is for both posting and replying the same, with the only difference being that the `PostBehaviour` quote-retweets instead of replying. They are both cascades of behaviours, checking for questions, lyric similarity and a default mispronunciation case. The bot first checks if profanities are used in the tweet, and if so, ignores the tweet and ends the interaction. The bot will then check if the tweet is a yes/no question by checking if the tweet ends with a question mark and does not contain a question word. If so, it reacts in Samson's stereotypical way of talking with a vague, 8-ball inspired answer, e.g. *"Mwaah zeg he, ik denk zo stil in mijn hoofd van wel zo ja."* (meaning *"But hey, I'm thinking quietly in my head of yes."*) It then checks if the tweet contains significant overlap with lyrics of one of the Samson & Gert songs by using n-grams, and if so, replies with a tweet containing part of the chorus, but with one of the last words of a line replaced by a rhyme, using a Dutch rhyming dictionary. The chorus is detected by looking for *"chorus"* markers in the lyrics, and otherwise taking a paragraph encompassing the most frequent

[7] https://twitter.com/SamsonRobot. Code: https://github.com/twinters/samson-bot

lines of the song. If the tweet does not either contain a question nor a lyric, it will mispronounce the longest word of the tweet, and ask what it means. It will also mispronounce the name of the Twitter user, and thanks to the deterministic mispronunciation search functionality, always use the same mispronunciation for the same user just like the real Samson does, which is usually appreciated by users of the bot[8].

4.2 GertBot

Gert is the other protagonist of Samson & Gert, and is the owner of Samson. In the show, he always corrects Samson when he asks what a certain word means, by repeating the word and then defining it.

Bot Behaviour We implemented GertBot[9] by looking up the word on Wiktionary[10], searching for the root word by recursively following the definition and shortening the final definition as much as possible. The bot also corrects the words that were incorrect whenever SamsonBot mispronounces lyrics. The `ReplyBehaviour` here is thus just listening to whenever SamsonBot tweets, and reacting on it immediately.

4.3 BurgemeesterBot

The Burgemeester is the mayor of the town of Samson & Gert and is known in the show for giving generic, inclusive speeches, e.g. *"Ahem, ahem, ahem. To everyone who came: congratulations. To everyone who didn't come, also congratulations"*. In order to implement this in BurgemeesterBot, we first needed a source of actions humans would do. The system also needs a method for negating actions to create the second part of the speech. These are then inserted into slots of the Babbly template, in which we were able to model several low probability hidden jokes into the speeches, to keep the bot more interesting.

Action Discovery The system uses titles of random WikiHow pages as a source of textual descriptions of possible actions, as this has been shown to be a good source of actions humans would generally do [18]. It also scrapes news headlines from a Belgian news site and with a lower probability uses actions retrieved from these headlines, as previous research showed how this makes Twitterbots feel more tied to current events [16,17]. It discovers possible actions in headlines by using a combination of the detailed POS-tags LanguageTools provides, and finds the most likely ones by using the probabilistic POS-tagger from OpenNLP. This way, it finds the most likely verb, and discovers words that are likely to be subjects, adverbs and other words establishing the action.

[8] As can be seen in this conversation: https://twitter.com/SamsonRobot/status/1001833743768793090

[9] https://twitter.com/Gert_bot. Code: https://github.com/twinters/gert-bot

[10] https://www.wiktionary.org/

Action Negation After the bot finds an appropriate action to speech about (either from WikiHow or from the news), it negates them for the second part of the speech. It does this by looking up antonyms for every word in the action on Wiktionary, and replacing one of the words with its found antonym. If no antonyms are found, it uses several hard-coded replacements, such a changing *"with"* to *"without"* and *"a"* to *"no"*

Bot Behaviour For the posting behaviour, BurgemeesterBot[11] selects a random Wikihow page[12] that does not contain a named entity, or scrapes an action from the news. When replying to a tweet, it uses the WikiHow search functionality on the tweet text, and keeps removing the shortest word from the tweet text until the search results in a page, in order to find a related human action. It uses this action along with the action negation system to fill in the slots in the Babbly template. This then might lead to the following tweet[13]: *"Aheuum. Aheuuuum. Aheum. Aan allen die luiheid overwinnen: proficiat. Aan allen die werklust overwinnen: ook proficiat."* (= *"Ahem. Ahem. Ahem. To all who overcome laziness: congratulations. To all who overcome work desire: also congratulations."*).

4.4 AlbertoBot

Albert Vermeersch is the food-obsessed hairdresser of the town of Samson & Gert. He is known for eating all of the food in the house of Samson and Gert, and constantly asking for more candy, cookies and other snacks. He is also known for correcting anyone calling him by his real name (Albert) to "Alberto".

Bot Behaviour The first behaviour the AlbertoBot[14] does is automatically following back anyone following him, and unfollowing them if they unfollow. This way, users (or other bots, for that matter) can indicate that they like the bot to occasionally interact with them. The first, small reply behaviour the bot has, is that it will correct anyone calling him "Albert" with his catchprase *"First of all, it's AL-BER-TOOO, second of all..."*, followed by some arbitrary statement. The main reply behaviour of this bot is that for every tweet on its timeline, it looks up which words can be food by searching them on Smulweb[15], a Dutch cooking website. If it finds enough evidence for a part of the tweet text to refer to food, he uses one of the many Babbly templates to show his enthusiasm for this food.

[11] https://twitter.com/BurgemeesterBot. Code: https://github.com/twinters/burgemeester-bot
[12] Using WikiHow's random page functionality https://nl.wikihow.com/Speciaal:Randomizer
[13] https://twitter.com/BurgemeesterBot/status/1006129059636621312
[14] https://twitter.com/AlbertBot. Code: https://github.com/twinters/alberto-bot
[15] https://www.smulweb.nl

4.5 OctaafBot

Octaaf De Bolle is the local shopkeeper of the show and is known for bragging about being a specialist in most actions that any other character mentions. For example, if someone were to mention creating Twitterbots, he might reply with *"Ah, creating Twitterbots! Now that happens to be one of my many talents. My daughter can confirm that, as she usually says: "Dad, the way you create Twitterbots...", well, that's how I create Twitterbots!"*. This is thus a simple template, which we can fill with any action found in a tweet using the same system BurgemeesterBot uses to scrape actions from the news, but instead from the tweets from the streams the bot listens to.

Bot Behaviour OctaafBot[16] has a reply behaviour as well as an identical post behaviour, which just quote retweets instead of replying. He listens to his mentions, his timeline and people mentioning his full character name on Twitter. Every so often, he selects a tweet that contains an action, and replies to it by bragging about his skills in this action. He also has a reply behaviour designed for tweets of his Twitterbot mom (JeannineBot), which makes him reply with *"Yes, mom, Yes"* whenever she gives advice, in which she usually undermines his statements.

4.6 JeannineBot

Jeannine De Bolle is Octaaf's mother in the TV show. She is usually very proud of being a handy person as the president of a local hobby club, and gives advice to all other characters. To generate appropriate advice for any tweet, we turn again to the WikiHow article search functionality and use the tips section at the bottom of related WikiHow articles, as they tend to include useful or sweet advice, like a mother would give. This advice is then filled into a Babbly template, which includes multiple ways her character might speak to people when giving the advice.

Bot Behaviour JeannineBot[17] has a reply behaviour that will always give advice to OctaafBot whenever he brags, and to other tweets from her timeline and mentions every so often. The other reply behaviour she uses on OctaafBot is continuing his brag by saying that he is so good at it because of her, or saying that she dislikes his brag and claiming it is something he got from his (absent) dad. An example of her advice can be seen on Figure 5.

[16] https://twitter.com/OctaafBot. Code: https://github.com/twinters/octaaf-bot
[17] https://twitter.com/JeannineBot. Code: https://github.com/twinters/jeannine-bot
[18] https://twitter.com/JeannineBot/status/1009515505290539010

NewsTweet:	"May moet uitgaan van mislukken Brexit-onderhandelingen" *("May must assume Brexit negotiations fail.")*
SamsonBot:	Whaa De Standaard, ik weet niet zo goed wat dat is, een "mislukten" zo. *("Whaa De Standaard, I am not sure what that is, a "failed".")*
OctaafBot:	Ah, mislukken! Dat is nu toevallig één van mijn specialiteiten! Mijn Miranda zegt dat ook altijd: "Pa," zegt ze, "zoals jij kan mislukken..." ja zo misluk ik hé! *("Ah, failing! Now that happens to be one of my specialties! My Miranda always says that too: "Dad," she says, "like you can fail..." Well, that's how I fail, you know!")*
JeannineBot:	Je weet het, mijn jongen: houd van jezelf. *("Remember, my boy: love yourself.")*

Fig. 5. An example of JeannineBot giving advice to OctaafBot[18]

5 Findings

5.1 Interactivity

All mentioned bots, except for GertBot, have general reply functionalities that can be used on any other tweet from the other bots, as well as any input from other Twitter users or even other Twitterbots. This usually causes interactive chains between the bots themselves, as well as other Twitter users.

Table 1. Summary of the start month, the followers and the average user interactions per tweet ($= \frac{likes+retweets}{\#tweets}$) on the posts and the replies of all bots, as of the 3th of September 2019.

Bot	Launch	# Followers	# Posts	# Replies	Avg. Post Int.	Avg. Reply Int.
`SamsonBot`	10/2017	180	2105	569	0,3145	0,2373
`GertBot`	10/2017	141	X	2647	X	0,2014
`BurgemeesterBot`	03/2018	111	1097	246	0,3081	0,2927
`AlbertoBot`	05/2018	106	X	1492	X	0,4068
`OctaafBot`	06/2018	75	612	95	0,1471	0,2211
`JeannineBot`	06/2018	63	X	970	X	0,1361

The number of followers, tweets and replies as well as the average number of interactions on posts and replies for these bots are summarised in Table 1. We can see that the two protagonists (SamsonBot & GertBot) are the most popular Twitterbots in their number of followers, even when taking into account the fact that they have had more time to gain more followers. AlbertoBot has the highest rate of interaction on its tweets. This might be thanks to his behaviour of following every Twitter user that decides to follow him, and from then on interact with them when they mention food, giving a notification to this user of his reply. SamsonBot and BurgemeesterBot have the highest interaction rate of interactions on their posts. This might be due to the fact that Samson has a wide range of complex behaviours, and his mispronunciations giving rise to humorous combinations of words. Another explanation might be similar to the high interaction rates of AlbertoBot, namely due to the fact that SamsonBot often quote retweets users tweeting about Samson & Gert, and thus giving a notification to the poster of the original tweet. For BurgemeesterBot, the inherent absurdism of some of the inverted actions in the speeches might explain his higher Twitter interactions. The lower OctaafBot interactions could be explained by the lack of variation that his typical bragging sentence has, especially given that the text in the slots is relatively less text than in BurgemeesterBot's template. This might also be the reason that JeannineBot has such low reply interactions, as her tweets are usually lengthy due to long template sentences, and long advice texts from WikiHow.

5.2 Examples

An example of a conversation including most of the discussed bots can be seen in Figure 6. Several Twitter users often like provoking the bot by either tagging them in conversations, or talking about food while being followed by AlbertoBot, as depicted in Figure 7.

6 Future Work

6.1 Babbly

There are several extensions that would greatly benefit the further development of Babbly. First, adding some form of tabling, relations or constraints would be incredibly useful to make sure that records are following additional constraints, such as respecting pronouns, or other constraints that must hold within a text such as alliterations or rhymes. Second, it would be interesting to add more analytical inference methods for better understanding the generative process, as has recently been done in other generative processes [5,7]. Currently Babbly supports calculating the probability of every option, and calculating how many different templates are encoded. Adding more inference methods might enable it to set several parameters in a more intelligent way, allowing users a more declarative programming approach. Third, it might help to revise the syntax of the language to be more like other frameworks, or more Pythonic, as this might help decrease the learning curve for new users. Fourth, it might be interesting to extend AIML with Babbly to provide a better chatbot framework that allows for more variation in its responses.

[19] https://twitter.com/BurgemeesterBot/status/1009539185299148800
[21] https://twitter.com/AlbertBot/status/1093079877824233472

TwitterUser:	LOL! Wie @SamsonRobot heeft geprogrammeerd is echt geniaal! *(“LOL! Whoever programmed @SamsonRobot is truly a genius”)*
SamsonBot:	Mwoh meneer Recht Ite, ik weet niet zo goed wat dat is, een "de programmeurd" zo. *(“Mwoh Mr. Straight Ite, I don’t know very well what that is, a "de programmerd" like that.”)*
GertBot:	Maar neen Samson, "geprogrammeerd"! Dat is het schrijven van een computerprogramma. *(“But no Samson, "programmed"! That is writing a computer program.”)*
OctaafBot:	Ah, het schrijven van een computerprogramma zijn! Dat is nu toevallig één van mijn specialiteiten! Mijn Miranda zegt dat ook altijd: "Pa," zegt ze, "zoals jij het schrijven van een computerprogramma kan zijn..." ja zo ben ik het schrijven van een computerprogramma hé! *(“Ah, being writing a computer program! Now that happens to be one of my specialties! My Miranda always says that too: "Dad," she says, "the way you can be writing a computer program ..." Yes, that’s how I’m the writing of a computer program!”)*
JeannineBot:	Je weet wat ik je heb gezegd, hé! Plan je volgende evaluatie met je leidinggevende door specifieke doelen te stellen en vast te leggen hoe deze doelen geëvalueerd worden. *(“You know what I told you! Plan your next evaluation with your supervisor by setting specific goals and recording how these goals are evaluated.”)*
BurgemeesterBot:	Ah! Toevallig heb ik nog een toespraak liggen over leidinggevende! Het gaat als volgt: Aheuum. Aheuuuum. Aheum. Aan allen die je sterke en zwakke punten bepalen: proficiat. Aan allen die niet je sterke en zwakke punten bepalen: ook proficiat. *(“Ah! Coincidentally, I still have a speech about a supervisor! It goes as follows: Aheuum. Aheuuuum. Aheum. To all who determine your strengths and weaknesses: congratulations. To all who do not determine your strengths and weaknesses: also congratulations.”)*

Fig. 6. An example Twitter conversation between the bots[19]

TwitterUser:	als niemand reageert, is daar nog altijd albertbot.... (aardappelgratin) *(“If no one responds, there is always albertbot .... (potato gratin)”)*
AlbertoBot:	Oooh mjammieee! Aardappelgratin! Ik hou zo enorm van aardappelgratin! *(“Oooh yum! Potato gratin! I love potato gratin so much!”)*
JeannineBot:	Ah, ik heb een hobbyclubtip daarover! Kies ervoor om van het leven te houden. *(“Ah, I have a hobby club tip about that! Choose to love life.”)*

Fig. 7. Twitter user provoking AlbertoBot using food, followed up by a reply from JeannineBot[21]

6.2 Twitterbots

The Twitterbots themselves could be improved by incorporating some other frameworks for different purposes. First, it might be useful to implement more behaviours in these bots, possibly relating more to building an actual plot, e.g. by using plot generators [10,11]. This would however decrease the independence of each bot, since they would most likely need some central system other than Twitter to communicate about the overall plot plans. Second, a useful upgrade to the system would be learning what type of tweets are most interacted with by Twitter users, and tweak its generative process to generate more of these types of tweets [19,20].

7 Conclusion

In this paper, we introduced a new text generation language called Babbly offering tight and elegant control over the probabilities of the templates, and by extend over the generated texts. We also created a framework for easily modelling complex Twitterbot behaviours, namely their post behaviour and their reply behaviour and the streams of tweets they are listening to. Afterwards, we demonstrated their use in modelling fictional characters as conversational agents by creating a complex network of six independent, conversational multi-agent system of Twitterbots. We then showed that these new bots are appreciated by their users. This indicates that these new frameworks can be valuable tools for other creators of conversational agents, especially in multi-agent settings.

Acknowledgements

This work was partially supported by Research foundation - Flanders (project G.0428.15).

References

1. Abdul-Kader, S.A., Woods, J.: Survey on chatbot design techniques in speech conversation systems. International Journal of Advanced Computer Science and Applications **6**(7) (2015)
2. Brandtzaeg, P.B., Følstad, A.: Why people use chatbots. In: International Conference on Internet Science. pp. 377–392. Springer (2017)
3. Colton, S., Charnley, J.W., Pease, A.: Computational creativity theory: The face and idea descriptive models. In: ICCC. pp. 90–95 (2011)
4. Compton, K., Kybartas, B., Mateas, M.: Tracery: An author-focused generative text tool. In: Schoenau-Fog, H., Bruni, L.E., Louchart, S., Baceviciute, S. (eds.) Interactive Storytelling. pp. 154–161. Springer International Publishing, Cham (2015)
5. Cook, M., Colton, S., Gow, J., Smith, G.: General analytical techniques for parameter-based procedural content generators. In: 2019 IEEE Conference on Computational Intelligence and Games (CIG). pp. 1–8. IEEE (2019)

6. Cook, M., Colton, S., Pease, A., Llano, M.T.: Framing in computational creativity–a survey and taxonomy. Proceedings of the 10th International Conference on Computational Creativity pp. 156–163 (2019)
7. Cook, M., Gow, J., Colton, S.: Danesh: Helping bridge the gap between procedural generators and their output. Proc. PCG Workshop (2016)
8. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. In: Soviet physics doklady. vol. 10, pp. 707–710 (1966)
9. Navarro, G.: A guided tour to approximate string matching. ACM computing surveys (CSUR) **33**(1), 31–88 (2001)
10. Riedl, M.O., Young, R.M.: Narrative planning: Balancing plot and character. Journal of Artificial Intelligence Research **39**, 217–268 (2010)
11. Riedl, M.O., Bulitko, V.: Interactive narrative: An intelligent systems approach. Ai Magazine **34**(1), 67–67 (2013)
12. Shawar, B.A., Atwell, E.: Different measurements metrics to evaluate a chatbot system. In: Proceedings of the Workshop on Bridging the Gap: Academic and Industrial Research in Dialog Technologies. pp. 89–96. NAACL-HLT-Dialog '07, Association for Computational Linguistics, Stroudsburg, PA, USA (2007), http://dl.acm.org/citation.cfm?id=1556328.1556341
13. Turing, A.M.: Computing machinery and intelligence. Mind **59**(236), 433–460 (1950), http://www.jstor.org/stable/2251299
14. Veale, T., Cook, M.: Twitterbots. MIT Press (2018)
15. Wallace, R.S.: The anatomy of alice. In: Parsing the Turing Test, pp. 181–210. Springer (2009)
16. Winters, T.: Generating philosophical statements using interpolated markov models and dynamic templates. In: 31st European Summer School in Logic, Language and Information Student Session Proceedings. pp. 181–189. Riga, Latvia, ESSLLI (Aug 2019)
17. Winters, T.: Generating dutch punning riddles about current affairs. 29th Meeting of Computational Linguistics in the Netherlands (CLIN 2019): Book of Abstracts (01 2019)
18. Winters, T., Mathewson, K.W.: Automatically generating engaging presentation slide decks. In: Computational Intelligence in Music, Sound, Art and Design - 8th International Conference, EvoMUSART. Ekart, A., Lecture Notes in Computer Science. Springer, Cham (2019)
19. Winters, T., Nys, V., De Schreye, D.: Automatic joke generation: Learning humor from examples. In: Distributed, Ambient and Pervasive Interactions: Technologies and Contexts. vol. 10922 LNCS, pp. 360–377. Streitz, Norbert, Springer International Publishing (2018)
20. Winters, T., Nys, V., De Schreye, D.: Towards a general framework for humor generation from rated examples. Proceedings of the 10th International Conference on Computational Creativity pp. 274–281 (2019)