Representations and Appropriate Simplifications of Data Accesses With Natural Language Generation

Leon Brooks
Seminar Inverse Transparency (WS 20/21)
Advisor: Valentin Zieglmeier
leon.brooks@tum.de

ABSTRACT

Data protection and privacy concerns are as relevant as ever and a topic that may significantly shape the future of our society. Companies and consumers alike have realized the value of data, although many people demand more insight on how their own data is used. Inverse Transparency offers a solution to relieve these tensions by informing owners when and how their data was accessed. This paper focuses on implementing Inverse Transparency through natural language generation systems to increase the accessibility and understanding for ordinary users. A simple example is conceptualized and implemented using the Jira REST API to modulate data accesses. Finally approaches to evaluating readability aspects are discussed.

KEYWORDS

Inverse Transparency, natural language generation, data privacy, readability, Jira

1 INTRODUCTION

In recent times the importance of data as valuable good has increased and most people are aware of this development [12, pp. 10 sq.]. At the same time around 50–60% feel their data is not safe online [12, pp. 6 sq.] [3, pp. 22 sq.]. People do care however what happens with their data [3, pp. 24 sq.] and over 80% expect companies to inform them how their data is used. Some would even shy away from using a product where they do not understand how data is used [4]. This has given rise to the concept of Inverse Transparency which demands transparency to owners over when and how their data is used for each individual data access [7]. It is based on the non-fiction book "The Transparent Society" by the science fiction author David Brin [2], an extract of which first appeared in the magazine Wired in 1996 [1]. In it, Brin describes a future of "open access" in which cameras monitor every public space and citizens, government and companies alike can freely monitor ones every move. As one solution for problems that may arise from this he suggests the "mutual-transparency solution" in which the system tells every monitored person who is watching them.

A problem that arises when implementing the more modern concept of Inverse Transparency is that the plain meta data, represented through access logs, may not be the optimal way to display data accesses to data owners. Therefore it is essential that the meta data is presented in a way that is easily readable and can be understood by ordinary people for Inverse Transparency to function as intended.

This paper will focus on ways to implement Inverse Transparency by displaying meta data as humanly readable sentences through natural language generation (NLG) systems. The applicability of readability metrics as well as the process of building a natural language generation system will be covered and implementation examples will be shown.

2 MOTIVATION

As touched on above, the primary intent of this paper stems from the question how to properly implement Inverse Transparency through an automated system in a form that is able to be understood by ordinary data owners. This should suggest a solution to the problem that true transparency can only be realized if every data owner can actually understand what is happening and in which way their data is used on a conceptual level instead of only being presented unrefined meta data.

In this paper the focus shall lie on giving an exemplary implementation of such a system and exploring questions that arise along the way of that process. This should build a foundation for improving and expanding Inverse Transparency systems and possibly encourage further research on the topic.

3 CONCEPTUALIZATION

This section shall focus on the requirements and specifications the exemplary implementation for this paper is supposed to fulfill. The relevance and basic functionality of the Jira REST API is shown and the usage in this system explained. Furthermore it is discussed which types of data accesses can and can not be detected given the specification and how these tie in with Jira.

3.1 Introduction and Usage of the Jira Rest API

Before one can build a NLG-system for an Inverse Transparency solution an input data set which describes the actual instances of user data being accessed is required. For this project the Jira REST API is used to implicitly define such a data set.

Jira is is a software system developed by the Australian company Atlassian which is used to manage agile software development life cycles such as Scrum.[1] Internally the software represents development tasks and goals with so called "Issues". They hold relevant meta data such as due date, assignee, time spent, etc. as well as user created remarks and comments.[2] Jira fits well as an implementation example as it is a platform which while performing it's primary purpose, providing an organization structure for agile software development, also provides the data for managers to monitor their subordinates work and performance. This is a practical scenario in which Inverse Transparency may help reduce tensions that arise

[1] https://www.atlassian.com/software/jira
[2] https://developer.atlassian.com/server/jira/platform/attachments/jira-7-9-2-database-schema.pdf

from the increase in availability of performance related data in the workplace [7, pp. 10 sqq.].

The Jira representational state transfer (REST) API can be used to interact with a Jira database via the HTTP protocol to allow third party developers to integrate a Jira database into their own software system. To access data one must generate a HTTP request which contains a search query for a specific or a group of issues[3]. One then receives a HTTP response containing a JSON representation of the requested issues. A request like this;

https://jira.atlassian.com/rest/api/latest/issue/JRA-9

could be used to retrieve the issue "JRA-9" from Atlassian's public issue tracker.

This project will use Jira REST API requests to represent instances of data access. Consequently the assignee, reporter and creator of a Jira issue are representing the data owners in the Inverse Transparency context. On a more concrete level the NLG-system will receive tuples of a HTTP request link and an accessing person as input. Then the requests will be analyzed and one or multiple sentences will be formed to express which data was accessed and what noteworthy patterns were detected therein.

It must be stressed that the system will neither implement the execution of the API requests nor generate the output from the hypothetical received data, but rather analyze the structure of the API request to form the resulting sentences.

Besides simply requesting a specific issue a request can instead contain a search query made of a Jira Query Language[4][5] (JQL) term. This would then return a collection of issues fulfilling the search query instead. A JQL term is built by combining search reference fields[6] of a Jira issue with specific operators[7] and search values. Since the scope of JQL terms far exceeds those relevant to realizing the Inverse Transparency concept the allowed input will be limited to the following ones:

- field = user
- field != user
- field IN (user,user,user,...)
- field NOT IN (user,user,user,...)
- field WAS user [options]
- field WAS NOT user [options]
- status WAS "Resolved" BY user [options]
- status WAS NOT "Resolved" BY user [options]

with field being either assignee, reporter or creator and user being a username, ID or e-mail address. [options] indicates optional date related clauses of the WAS operator (AFTER, BEFORE, ON, DURING). A request like this for example;

https://jira.atlassian.com/rest/api/latest/search?
jql= status WAS Resolved BY "mike@atlassian.com"

would return all issues that were resolved by the user with the e-mail address "mike@atlassian.com".

[3]https://developer.atlassian.com/server/jira/platform/rest-apis/
[4]https://www.atlassian.com/software/jira/guides/expand-jira/jql
[5]https://3kllhk1ibq34qk6sp3bhtox1-wpengine.netdna-ssl.com/wp-content/uploads/2017/12/atlassian-jql-cheat-sheet-2.pdf
[6]https://support.atlassian.com/jira-software-cloud/docs/advanced-search-reference-jql-fields/
[7]https://support.atlassian.com/jira-software-cloud/docs/advanced-search-reference-jql-operators/

For these types of requests it arguably appears to be more fitting to only notify the user(s) targeted within the search queries as only they are part of the intent of the data access. For example if a superior searches for all issues resolved by a certain user it does not make sense to notify the creator and reporter of every issue as well.

3.2 Detecting Relevant Patterns Within Instances of Data Access

Simply notifying the user when and which of his data has been accessed may be an important task of a software system implementing Inverse Transparency, but on its own arguably does not add significantly more value than simply displaying logs. Therefore, one must ask themselves which types of access behaviours may be of special interest to data owners, how they can be determined from the access data and in which way the NLG-system should emphasize them.

Since no literature or surveys on this specific question could be found some plausible examples were chosen. These include a single person accessing a large amount of data from a specific owner, a person accessing certain data on a regular basis/in fixed intervals or a specific set of data being accessed by many users.

Before applying these patterns to the system however, one must address the notion of time first. As mentioned in 3.1 the system shall receive tuples of the accessing person and the Jira API request as input. This poses a problem since it would not be sufficient to realize a time frame based analysis. On the other hand implementing timestamp parsing and a calendar based tracking or real time notification system is beyond the scope of this simple system, as the main focus of this project lies on the NLG component.

To solve this discrepancy it shall be defined that the input contains all access data of a certain fixed time frame. This could be a week, a month or an arbitrary number of days. Since the concrete time frame matters for the NLG, it will be given as a startup parameter. Intuitively this should imply that the data accesses (i.e. the Jira API requests) are tracked externally and the system is then run every week/month/etc with the access data as Input.

At this point it makes sense to specify the output format as well. Similarly to the input the output will consist of tuples of a natural language message and the intended recipient.

With these specifications complete, the question remains which of the patterns mentioned above can be detected within this system and how they can be derived from the input data.

A person accessing large amounts of data from a specific owner can be detected very easily by simply counting the =, IN and WAS operator calls. If they pass a certain predetermined threshold they are considered noteworthy.

Detecting a person accessing data on a regular basis is not possible however, given the specification set above. Without concrete timestamps or a database storing of results over multiple runs of the application this is not feasible.

As with the previous pattern, a specific data set being accessed by many users is not detectable since only the queries, not the actual issues returned by them, are known. A similar goal however can be achieved by tracking how many users searched for a specific owner using the =, IN and WAS operators.

Besides the pattern based approach it might be of interest to an owner if a rare request occured or a request which may imply a certain intent was issued. All negated operators (!=,NOT IN, WAS NOT) should represent the first category as it is arguably less common to search which issues a person was not involved in or did not resolve. The second category can be represented by queries including the "Resolved" BY clause or request with date specific options. These request are, amongst other purposes, also used when monitoring responsibility, accountability and productivity. Therefore they will justify a separate notification in this system.

4 IMPLEMENTATION

This section will show, step by step, the concrete realization of the requirements and functionalities specified in 3. At the same time it will be shown how this implementation relates to common NLG design concepts.

The programming language chosen for this project is Java. This is due to the rich functionality of the Java String class and the simple concatenation through the + operator. The complete implementation is available on GitHub.[8]

4.1 Input and Output Format

A common practice for specifying requirements of a NLG-system is the corpus based approach [10, p. 5], where concrete input and output examples are used to define the functionality a NLG-system should have. In section 3 a similar thing was done on a more abstract level by defining the input and output tuples[9]. Here we want to further elaborate on the concrete form this will take in the implementation.

The program will take the input as a txt file, the path to which is given by a startup parameter, that holds the tuples. As a separator character between requester and request "{" will be used since it is neither part of the URI[10] nor the JQL alphabet. Tuples themselves will be separated through new lines. A valid line of the input file could look like this:

James{https://jira.atlassian.com/rest/api/latest/search?jql=creator = Peter

The output will be written to a txt file as well. For each user appearing in one or multiple search request one message will be generated. The message is a summary of all notable accesses detected within the input and may consist of multiple sentences. The exact format will not be further specified.

4.2 System Components and Class Structure

In their Book "Building Natural Language Generation Systems" [11] Ehud Reiter and Robert Dale describe a popular model for building NLG-systems. Although not without criticism [9, pp. 3 sqq.], it "Arguably [...] is still the most complete available survey of NLG" [6, p. 5]. It describes six basic tasks a NLG-system fulfills to get from input data to a finished text:

1. *Content Determination:* choosing which parts of the input data will appear in text later
2. *Discourse Planning:* ordering/giving a structure to the items created in the previous task
3. *Sentence Aggregation:* deciding which items will be part of the same sentence
4. *Lexicalization:* deciding which words to choose to express the information the items represent
5. *Referring Expression Generation:* choosing pronouns or phrases which can be used to refer to items to increase fluidity of the final text (e.g.: it, her, himself, etc.)
6. *Linguistic Realisation:* the final step of forming correct sentences from the chosen words

These tasks are then distributed over three different modules (see Figure 1) used to implement NLG-systems. The Text Planner is responsible for tasks 1 and 2, the Sentence Planner fulfills 3,4 and 5 while the Linguistic Realiser implements task 6.

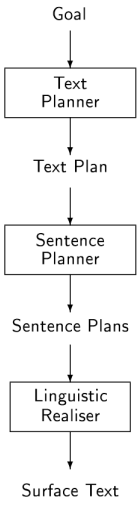


Figure 1: Figure by Reiter and Dale [10] explaining their NLG-system architecture model. Boxes stand for modules while text stands for the output of each module

Since this application is rather simple and fairly static in its outputs, a sophisticated structure like this is not required and would only over complicate the implementation with little benefit. It is a very useful guideline however to help formulating ones own approach to building a NLG-system.

When for example asking the question: "How and where would Content Determination appear in this system", one realizes that this would most likely fit the counting and dissection of the search request for each user. More specifically this would mean iterating over all search requests and keeping track of how often each user was targeted in each different type of request.

Once this data is complete one could continue with the other tasks. For our system however it is clear that in the end a few similar sentences combined with variables can be used to derive all possible output messages. This is a case where a template based approach appears to be a well suited option [6, p. 16]. Templates can be summarized to be predefined sentences including placeholder variables. They can usually look somewhat like this:

[8] https://github.com/LeonBrooks/Inverse-Transparency-NLG
[9] in: (requester, HTTP Jira REST API request), out: (recipient, data acess description message)
[10] as defined in RFC 3986: https://tools.ietf.org/html/rfc3986

(1) Your colleague *$requester* searched for you *$count* times.

The main task of the system is then to choose the sentences and set the variables according to the precomputed data.

This then suggests a two module architecture for this system. An `InputAnalyzer` which reads the input and creates a data set for further processing and a `TextGenerator` that takes this data and compiles the output. Therefore the Content Determination task is fulfilled by the `InputAnalyzer` while the remaining tasks are handled by the `TextGenerator`. The Discourse Planning, Sentence Aggregation and Lexicalization task although are strictly speaking not actively performed but rather implicitly given by the template setup.

4.3 Concrete Implementation Details

Besides the `InputAnalyzer` and `TextGenerator` modules the complete program will also include two more classes. A Main class which contains the main method that reads from the input file and writes to the output file. As arguments the main method takes:

- The path to the input file
- The path to the output file
- A string that describes the time frame (e.g. "week", "month", "15 days")
- An integer `threshold` that is the maximum number of requests issued by the same user, which are considered not noteworthy
- An integer `multiuserThreshold` that is the maximum number of requesters querying for the same users, which are considered not noteworthy
- an optional string "detailed", which will cause the program to print exact numbers and listings

The `RequestExtract` class functions as a data object that contains all the relevant metrics needed to properly navigate the templates within the `TextGenerator` class. One `RequestExtract` object represents all the data needed to generate a message for an individual data owner. It has five attributes of which the first two contain the total amount of request made targeting a user and the number of unique requesters, while the remaining three contain the data needed to realize the pattern detection specified in 3.2.

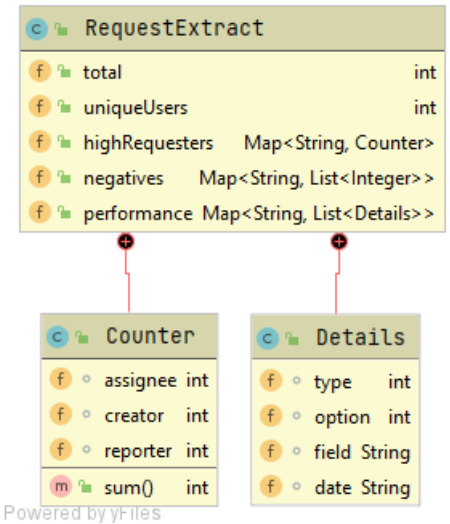


Figure 2: Class diagram of `RequestExtract`

The `highRequesters` attribute maps requesters whose request count exceeded the `threshold`, to an object of the inner class counter that tracks the queried field of all requests. Negatives maps requesters who made negative requests (!=, NOT IN, WAS NOT) to a list of integers from which the request type can be determined. And the performance attribute maps requesters that made request that could be related to tracking the performance of a user, to an object of the inner class Details that contains a complete breakdown of those request.

4.3.1 Input Analyzer. The `InputAnalyzer` class is responsible for converting the Jira REST API request strings into data objects the `TextGenerator` can easily interpret and use to generate the final messages. Therefore, it creates a `RequestExtract` object for every user who appeared in any request, which summarizes all requests that user was targeted by.

This is handled by the static method `analyzeInput()` which receives a list String[] and the `threshold` argument as input (see Figure 3). Each array has the size 2 and contains the requesting user and the request. It returns a Map associating each targeted user to their individual Request extract.

To accomplish this the method iterates over all requests and fills the output map accordingly. As a first step the request string is cut down to only the JQL portion (e.g. creator != Paul) and the request type is determined (one of the eight types defined in 3.1). Then the targeted users are identified and for each of them the corresponding `RequestExtract` is retrieved from the result map. If the user appeared for the first time a new `RequestExtract` is created.

The request is then categorized as performance related, negative or standard. The related checks are performed in that order. Performance related requests are those which query for WAS (NOT) "Resolved" BY or those which contain a date specific option. Negative request, as the name implies, contain a negated operator and all other request are classified as standard.

For performance related and negative requests, new items are inserted into the lists associated to the user who issued the request. For negative request this is only a single Integer which represents the negative request type.

This is not the standard request type ranging from 0–8 however. Since all negative requests query for a field (assignee, creator, reporter), otherwise they would be categorized as performance related, the only other difference is if they queried for a single (!=,WAS NOT) or for multiple users (NOT IN). Therefore, the negative request type is mapped to 0–5 as follows:

- type = 0: single user querying for assignee field
- type = 1: single user querying for reporter field
- type = 2: single user querying for creator field
- type = 3–5: multi user querying for assignee, report, creator

For all request the Counter object associated with the requesting user within the `highRequesters` attribute is updated accordingly

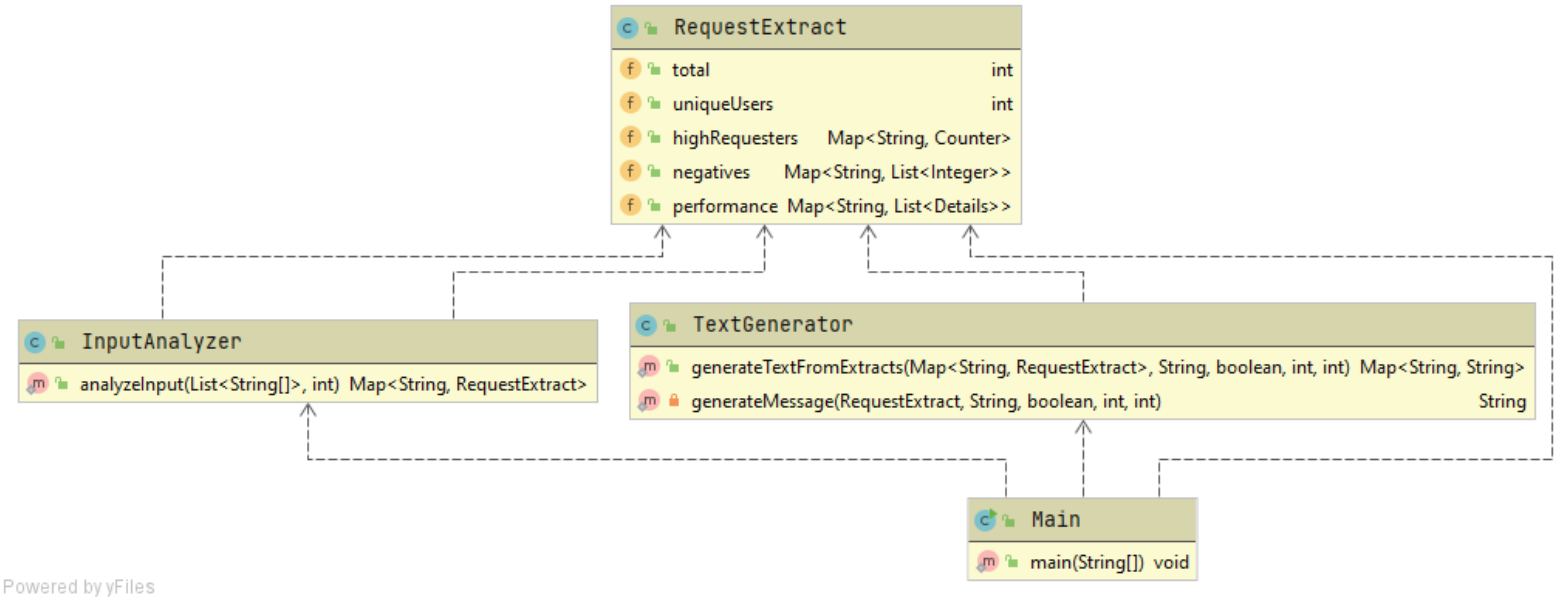


Figure 3: Class diagram of the complete system

(request querying for "Resolved" BY count towards the assignee field) and the total attribute is incremented.

Lastly after all requests have been processed the `uniqueUsers` attribute is set for every `RequestExtract` object. It can simply be set to the size of the `highRequesters` map as, at this point, every requester has an entry within it. Finally all requesters whose combined request counts (Counter.sum()) are below the `threshold` parameter are removed from the `highRequesters` map.

4.3.2 Text Generator. The `TextGenerator` class consist of the static method `generateTextFromExtracts()` which takes the map created by the `InputAnalyzer`, the `timeframe`, the `detailed` boolean, the `threshold` and the `multiuserThreshold` as parameters and returns a map associating each requested user with the message they are supposed to receive. The creation of each individual message is outsourced into the private `generateMessage()` method which takes a `RequestExtract` instead of the map and the same remaining parameters as the `generateTextFromExtracts()` method.

At first glance it may seem simple to build a template based NLG-system but in practice there are quite a few details to pay attention to. In this case the message generation is split into four major parts according to the patterns defined in 3.2 and represented by the `RequestExtract` attributes. Besides the overall decision making process of which patterns to include, for each part some grammar and logic related actions must be performed as part of the Linguistic Realisation task.

- *Agreement*: Some words need to be adjusted depending on other words they relate to. For example the words *colleague* and *is* must agree in number in the phrase: "[...] your colleagues were interested in [...]" (3-rd person plural). It can not be "[...] your colleagues was interested in [...]" (*colleague*: 3-rd person plural, *is*: 3-rd person singular).
- *Punctuation rules*: When enumerating for example, two people must be separated with an and, but for more people, the first x people must be separated by a comma while the last two should still be separated by an and.
- *different formulation according to predecessors*: To enhance the fluidity and readability of the result, each text block connected to a pattern will differ depending if previous patterns were detected or not.
  For example the second of the two blocks "Many people accessed your data this week. Specifically your colleague John was interested in [...]" will change to "Your colleague John was interested in [...] over the last week" if the pattern connected to the first was not detected.

Adjustments for all these factors are made in every block described in the following and, although arguably the most effort in the actual implementation, are not detailed further as this would most likely only be a list of questions similar to: "Does x word need a plural s appended in this, this and that case ?". For interested readers I refer to the complete implementation.[11]

As mentioned above the text is generated in four major blocks each connected to a pattern. The first is the relatively simple question, if many individual people made requests querying for the user in question. If this is the case the first sentence will be: "Many people accessed your data this week."

The next block is connected to the pattern if a singular user has a high request count. As mentioned in 4.3.1 every requester that fulfills this criteria has an entry in the `highRequesters` attribute of the `RequestExtract`. If applicable respect is paid to which field(s) they requested the most. This then generates sentences like: "Specifically your colleague John seemed very interested in which issues

[11] https://github.com/LeonBrooks/Inverse-Transparency-NLG

you created and were working on." or "Your colleagues Jill, Joe and Mike seemed very interested in your activity over the last week."

The third block is related to whether people made negated requests. It functions similarly to the previous one and produces output like: "Additionally your colleague Tara made requests a request from which they can deduce on which issues you were not the assignee." or "Your colleagues made requests from which they can deduce on which issues you were not the assignee, creator or reporter over the last week."

The last block is dedicated to possibly performance related requests. Here requesters are grouped into categories depending on what types of requests they made. These categories include requests that were querying for which issues an assignee did and did not resolve and requests that had a date options attached to them. This can produce sentences like: "Some of your colleagues also made requests which could be related to tracking your performance or responsibilities: Patrick and John were interested in which issues you resolved, Tina was interested in which issued you did not resolve and Tom, Bill and Jane checked if you were involved in an issue within a certain time frame."

Finally as touched on above the detailed options will cause additional information to be printed for each block. For the first that is the total number of requests. For the second this is a print out of the counter object for each "high requester". Similarly, for the negated requests the number of request of each requester are printed out by type. Lastly for the performance related request a breakdown of every individual request is given. Just like the remaining message, all of these are formulated in natural language as well.

5 EVALUATING THE READABILITY OF GENERATED LANGUAGE

One of the stated goals of this paper is to give an exemplary implementation of a NLG-System which may help ordinary users understand access mad to their data. For this to be achieved it must be assured that users actually understand the output of the system. Therefore one must find appropriate metrics to evaluate the generated messages.

5.1 Applicability of Classic Readability Metrics

At first it may seem appropriate to use methodologies from the field of readability studies. There are different definitions of readability but the work in this field may be summarized by the concern of ensuring a text affects the reader in the way the author intends [13, p. 1]. Many different metrics have been developed to evaluate the readability of a text and most use average sentence length and a form of word length (e.g syllables per word) as primary variables to determine the difficulty of a text [13, pp. 2 sqq.].

In the context of NLG-systems and this particular case they may seem unfit for three main reasons. First and foremost readability metrics were developed with the goal of evaluating longer texts like (text)books or newspaper articles [13]. For two popular formulas, the Flesch Reading Ease and Flesch-Kincaid Grade Level, Graesser et al. stated they could only be successfully applied for texts with more than 200 words [8, p. 7]. Another popular metric created by Edward Fry [5] uses three 100 word samples to determine an appropriate age of the reader. The output of this NLG-system only produces messages of about 100 words in total and although not explicitly stated for every readability formula, this seems not to be the intended type of text.

Secondly, the fact that readability metrics cannot make proper judgements on how well a text will be understood by the reader [13, p. 9] cannot be ignored. Although very similar intuitively, readability and understanding a text do not have exactly the same meaning in this context. The nuanced difference here is in the overall fluidity and understanding what the words of a text say versus understanding the meaning behind the text. As the latter is specifically what this system tries to achieve this makes them of little use in this case.

And finally it must be considered that as mentioned above classic readability metrics were developed to evaluate human authored text. Even if one would only want to evaluate the fluidity of generated text, metrics based on average sentence length, average word length and possibly even different list of easy or difficult words seem sub optimal. If, for example, the referring expression generation of a NLG-system were missing the text generated could look like this:

Tom went to the store to go shopping. There Tom bought apples, pears and some vegetables. Tom needed the fruit for Tom while the vegetables were for Tom's mother.

This is arguably less readable and fluid than:

Tom went to the store to go shopping. There he bought apples, pears and some vegetables. He needed the fruit for himself while the vegetables were for his mother.

Human authors would most likely never produce the first text. Therefore classic readability formulas based on the metrics mentioned above were not developed to detect such problems and would score both texts quite similarly.

For these reasons it seems inappropriate to use classic readability metrics to asses NLG-systems and especially in this case further research may be required to properly evaluate how well the generated messages convey and summarize the different data accesses.

5.2 Readability Survey

Although classic readability metrics are not applicable this paper does however aim to provide some form of readability evaluation. Therefor a public survey was performed. It must be stressed that the goal was not to produce a scientifically or statistically representative evaluation but rather to gather first insight into the quality of the generated output.

The survey was taken by 23 participants and had no restrictions concerning any types of prerequisites. The complete form including all questions and answers with diagrams can be found in the appendix (section A) as well as online (only answers).[12]

In total six questions were asked in reference to a single output message:

(1) Did participants have any previous experience with Jira
(2) How fluent and natural did the output sound
(3) How well did the participants think they understood the output
(4) How easy did they think the output was to understand in general

[12] https://docs.google.com/forms/d/e/1FAIpQLSfiAKzNv1pOcEVXhZK7Ic3JtFh-hkzi776_gctJfN1f320W-w/viewanalytics?usp=form_confirm

(5) How well did the message represent the queries it was generated from
(6) In what ways could the output be improved

The message was the output form accesses containing all four patterns described in sections 3 and 4. Before the questions were asked brief explanations of the concepts of Inverse Transparency, NLG and Jira were given. Evaluation questions were based on a scale of 1 to 4.

In total only 13% (3) of participants were familiar with Jira. The fluidity of the message appeared to be satisfactory with lowest score being 2 which was only given by 17% (4). A large majority of 65%(15) thought they understood the message completely and only one person, who gave a score of 2, had problems comprehending the output. Curiously the simplicity of the message in general was rated slightly worse. Only 39% (9) participants thought everyone could understand the message and a score of 2 was given by 26% (6) people here instead. This may indicate that most participants considered themselves to be better qualified to understand the message than the general public.

For the last evaluation question the participants were presented with all the queries which were used to generate the message they had been previously judging. In addition to grading how well the output summarized the request queries they had the option to not give an answer because they did not understand the request format. About half gave a score of 3, a quarter a score of 4 and the remaining quarter consisted of three people who did not understand the format and three people who gave a score of 2. The last question could be answered freely in text and participants made a wide variety of suggestions to improve the system. The full list of these can be found in the appendix as well.

Overall the system appears to fulfil its purpose of generating messages which can convey data accesses to a general public. Although there is definitively room for improvement, no participant gave a sore of 1 on any question and on every question at least 74% (17) gave a score of 3 or 4. Especially since all but three participants had no previous experience with Jira this can be seen as a satisfying result considering the stated purpose. Specifically even all three participants that did not understand the request format scored their own comprehension of the message at least a 3. This indicates the system might even be able to allow people who would not understand plain access logs to get an insight on how their data was used.

6 CONCLUSION

In this paper the basic concept of Inverse Transparency and how NLG-systems may help implement it was explained. A simple system based on analyzing accesses made through the Jira REST API was implemented. It was shown how and to which degree NLG design concepts are helpful when conceptualizing a basic system and how they translate to concrete implementation solutions. Lastly a short look was taken at why classic readability metrics may not be suited to evaluate NLG-system in general and what problems arise in this specific case. Instead a simple survey was conducted to gather a surface level understanding of the quality of the output. Its results indicate the system likely achieves its goal of giving ordinary people an insight into how their data is used.

This should help to set a stepping stone for further research on practical solutions in the field of Inverse Transparency. Especially the evaluation of implementations of this concept may require new solutions and research in this area should be strongly encouraged.

7 CONTRIBUTION

The paper aims to contribute to the topics of Inverse Transparency and natural language generation systems by giving an exemplary implementation and elaborating on common features, function and requirements of such systems. This practical approach to Inverse Transparency may help people unfamiliar with the topic to understand it's purpose and potentially serve as a foundation to build more complex and sophisticated systems.

REFERENCES

[1] David Brin. "The Transparent Society". In: *Wired* (1996). URL: https://www.wired.com/1996/12/fftransparent/ (visited on 02/05/2021).
[2] David Brin. *The Transparent Society*. New York City, US: Perseus Books, 1998.
[3] Telekommunikation und neue Medien e.V. Bundesverband Informationswirtschaft. *Datenschutz im Internet - Eine repräsentative Untersuchung zum Thema Daten im Internet aus Nutzersicht*. Albrechtstraße 10 A, 10117 Berlin-Mitte, 2011. URL: https://www.bitkom.org/sites/default/files/file/import/BITKOM-Publikation-Datenschutz-im-Internet.pdf (visited on 11/29/2020).
[4] Bundesverband Digitale Wirtschaft e.V. *Verbraucherumfrage zum Thema Datenschutz*. Schumannstraße 2, 10117 Berlin, 2019. URL: https://www.bvdw.org/fileadmin/user_upload/BVDW_Verbraucherumfrage_Datentransparenz.pdf (visited on 11/29/2020).
[5] Edward Fry. "A Readability Formula That Saves Time". In: *Journal of Reading* 11.7 (1968), pp. 513–578.
[6] Albert Gatt and Emiel Krahmer. "Survey of the State of the Art in Natural Language Generation: Core tasks, applications and evaluation". In: *Journal of Artificial Intelligence Research* 61 (2018), pp. 65–170.
[7] Maren Gierlich-Joas, Thomas Hess, and Rahild Neuburger. "More self-organization, more control—or even both? Inverse transparency as a digital leadership concept". In: *Business Research* 13 (1 2020), pp. 921–947.
[8] Arthur Graesser et al. "Coh-Metrix: Analysis of text on cohesion and language". In: *Behavior Research Methods, Instruments, Computers* 36.2 (2004), pp. 193–202.
[9] Chris Mellish et al. "A Reference Architecture for Natural Language Generation Systems". In: *Natural Language Engineering* 12.1 (2006), pp. 1–34.
[10] Ehud Reiter and Robert Dale. "Building applied natural language generation systems". In: *Natural Language Engineering* 3.1 (1997), pp. 57–87.
[11] Ehud Reiter and Robert Dale. *Building Natural Language Generation Systems*. Cambridge, UK: Cambridge University Press, 2000.
[12] Symantec. *State of Privacy Report 2015*. Found on statista.com yet the original source link to Symantec was invalid due to symantec.com being integrated broadcom.com, access link (nortonlifelock) was found over google.com. 2015. URL: https://www.nortonlifelock.com/content/dam/nortonlifelock/pdfs/reports/state-of-privacy-report-en-2015.pdf (visited on 11/29/2020).
[13] Mostafa Zamanian and Pooneh Heydari. "Readability of Texts: State of the Art". In: *Theory and Practice in Language Studies* 2.1 (2012), pp. 43–53.

A SURVEY QUESTIONS

Evaluating Generated Language in the Context of Inverse Transparency

(Since this survy was designed to be anonymous it does not require a log-in. Please only submit your answers once.)

Inverse Transparency is a privacy concept that allows data access but in return requires them to be transparent to data owners.

In other words data owners must be informed about when an access occurred and who accessed their data.

Natural language generation systems may help to make this process easy to understand for a larger public. They can convert the summary of data accesses into a simple text which can be displayed to data owners.

This survey focuses on evaluating how simple such a text really is and how well it represents the access data.

1. Do you have any previous experience with the software system Jira? (not required for this survey)

   Mark only one oval.

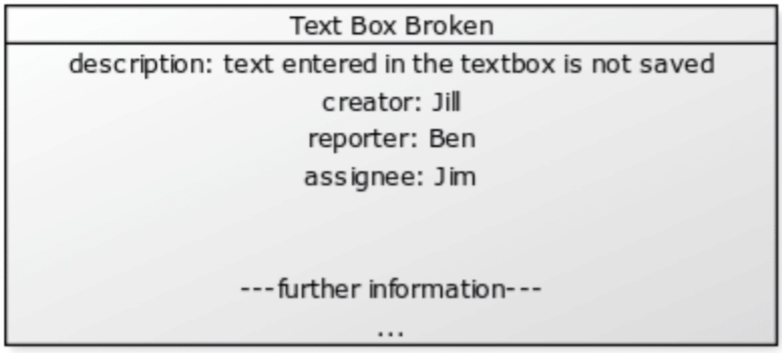
   - ◯ Yes
   - ◯ No

Jira

Jira is a software system from the Australian developer Atlassian. It is used to manage agile workflows such as Scrum.

Important for this survey is only that Jira stores tasks and problems in so called "issues". Besides other information an issue contains a Description of the problem and stores who created it, who reported the problem and who was assigned to solve it.

An issue could look something like this:



Accesses of Jira Issues

Our system now takes data from users searching within a database containing Jira issues. It then generates text to inform the people who were involved in the searched issues (creator, reporter and assignee) that their data was accessed.

Consider the following message:

Many people accessed your data within the last week. Specifically your colleague John seemed very interested in which issues you were working on.

Additionally your colleague John made requests from which they can deduce on which issues you were not the reporter.

Some of your colleagues also made requests which could be related to tracking your performance or responsibilities:
Patrick and John were interested in which issues you resolved, Tina was interested in which issues you did not resolve and John checked if you were involved in an issue within a certain time frame.

2. How fluent and natural does the message sound?

Mark only one oval.

	1	2	3	4	
the text sounds terrible	◯	◯	◯	◯	nothing hinders the readability

3. With the previous information, do you think you understand what the message is trying to say?

Mark only one oval.

	1	2	3	4	
I have no idea what the message is about	◯	◯	◯	◯	I understood everything

4. With the previous information, how easy do you think the message is to understand in general?

Mark only one oval.

	1	2	3	4	
you have to be an expert to understand it	◯	◯	◯	◯	everyone can understand it

Jira Search Requests

As mentioned previously the message was generated from search requests. The following were the input for this message. The name on the left before the colon is the requesting person. Then recipient of the Message is Jim.

```
John:    assignee     =       Jim
John:    assignee     IN      (Jim,Tara)
John:    reporter     NOT IN  (Jim,Tara)
John:    assignee     WAS     Jim              ON  1.1.2021
John:    status       WAS     "Resolved"       BY  Jim

Tina:    status       WAS NOT  "Resolved"      BY  Jim

Patrick: status       WAS      "Resolved"      BY  Jim
```

Resulting Message (Same as before):

Many people accessed your data within the last week. Specifically your colleague John seemed very interested in which issues you were working on.

Additionally your colleague John made requests from which they can deduce on which issues you were not the reporter.

Some of your colleagues also made requests which could be related to tracking your performance or responsibilities:
Patrick and John were interested in which issues you resolved, Tina was interested in which issues you did not resolve and John checked if you were involved in an issue within a certain time frame.

5. How well do you think the message summarizes the search requests?

Mark only one oval.

- I don't understand the request format
- terrible summary
- 2
- 3
- perfect summary

Improvements

6. What could be improved about the generated message?

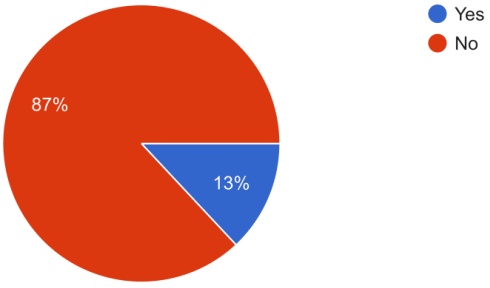
B SURVEY ANSWERS

Evaluating Generated Language in the Context of Inverse Transparency

23 responses

Do you have any previous experience with the software system Jira? (not required for this survey)

23 responses

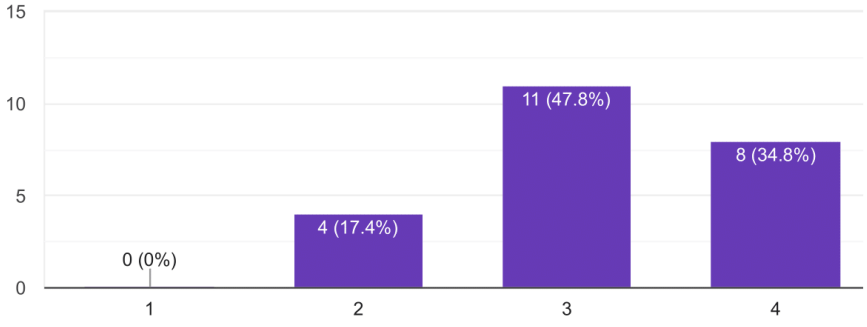


Jira

Accesses of Jira Issues

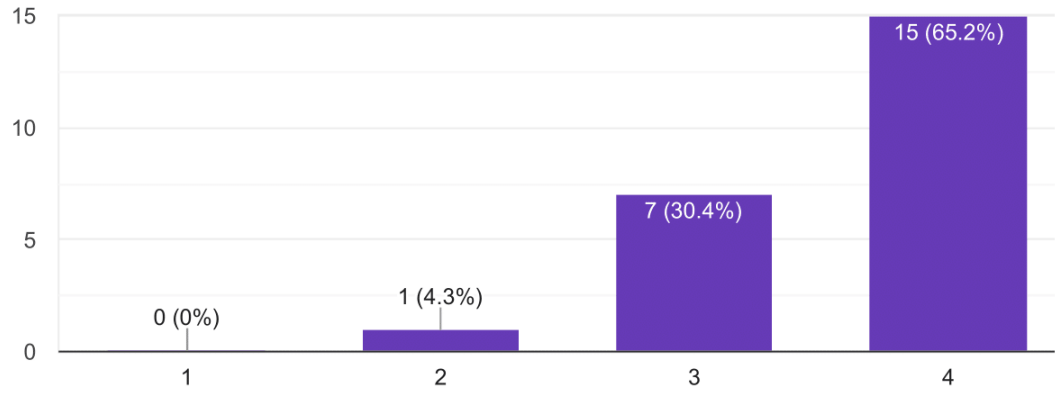
How fluent and natural does the message sound?

23 responses



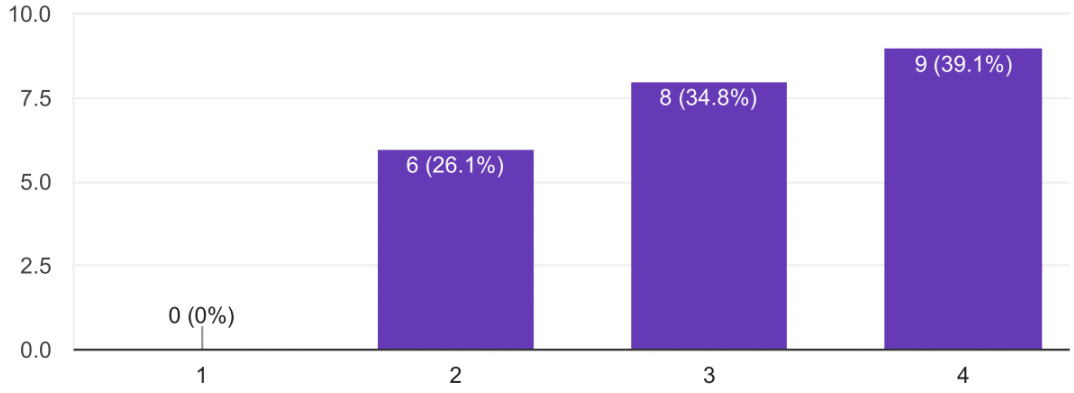
With the previous information, do you think you understand what the message is trying to say?

23 responses



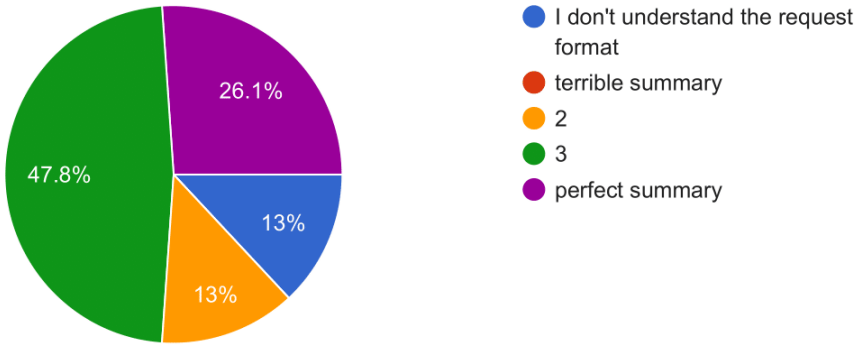
With the previous information, how easy do you think the message is to understand in general?

23 responses



How well do you think the message summarizes the search requests?

23 responses



What could be improved about the generated message?

11 responses

I don't really know, I guess some people could struggle with certain vocabulary like "deduce", maybe use conclude? Deduce also makes me think of Sherlock Holmes.

Options for different grouping of messages like grouping by names (John wanted to know...) or by accessed information etc. General summary was good

The negations could be changed: E.g. "issues you weren't the reporter" reads a bit smoother. Or maybe the sentence can be rewritten to "issues someone else was the reporter"?

I think the message could be better understood if the sentences would involve numbers and lists. Like: John access your data 5 times. He was interested in:
- if you were assigned
- if you were the reporter
- ...

It is functional, so i wouldn't change it

What could be improved about the generated message?

11 responses

It is functional, so i wouldn't change it

Do one paragraph per bit of information or get rid of paragraphs entirely. Replace "many" with an exact number.

I would prefer if the message would be sorted by "requesting person". To combat timeline confusion, the time and date of each request could be added.

I think some parts sound too "not humane" but that isn´t a big problem. Parts like "which isues" could be changed to "the issues that" but that is personal preference

bulletpoints instead of a whole paragraph -> more readable

"Your collegue XYZ" sounds artificial

shorter textblocks, maybe categorized for different persons (John appeared in two textblocks, maybe collapse all the queries john did into one textblock). Using bold text for important keywords could improve the readability (f.ex. "John" requested to see which issues were "not resolved" by "Jim")