Reasoning Engine for Support Maintenance

Rana Farah[1(✉)], Simon Hallé[1], Jiye Li[1], Freddy Lécué[1,3(✉)], Baptiste Abeloos[1], Dominique Perron[1], Juliette Mattioli[2], Pierre-Luc Gregoire[1], Sebastien Laroche[1], Michel Mercier[1], and Paul Cocaud[1]

[1] Thales Digital Solution, Québec, QC G1P4P5, Canada
rana.farah@polymtl.ca, {simon.halle,jiye.li,baptiste.abeloos, dominique.perron,pierre-luc.gregoire,sebastien.laroche, michel.mercier,paul.cocaud}@ca.thalesgroup.com, freddy.lecue@inria.fr

[2] Thales, 92098 Paris La défense, France
juliette.mattioli@thalesgroup.com

[3] Inria Sophia Antipolis, 06902 Valbonne, France

Abstract. This paper presents a reasoning system deployed for supporting the maintenance of IT devices in use by a leading broadcasting and cable television company in North America. We describe a reasoning engine pipeline relying on semantic data representation and some machine learning approaches such as clustering. The engine derives problems on a telecommunication network from a textual description and uses structured historical data of problems, error codes and proposed solutions to prescribe potential solutions. The engine is capable of proposing solutions to unseen problems by using analogical reasoning on structured representations. When a problem happens on the network or more precisely on one of the devices, these devices generate error codes. We addressed two scenarios; (i) we assumed that the list of error codes that we captured is complete, (ii) we assumed, more realistically, that this list is incomplete. In the first case, we suggested solutions for seen and new problems and reported results on real data. In the second case, we proposed a method to infer the complete list of errors, tested that method on synthetic data and showed results with high accuracy. Although both scenarios are in-use, the first scenario is more usual than the second one, but both need to be considered.

Keywords: Knowledge graphs · Graph embedding · Reasoning engine

1 Introduction and Literature Review

Facilities-based providers are looking into tools that preempt problems on their service platform, network and their customers' equipment. To support their customers, they employ a huge amount of resources being the Network Operation Centers (NOCs), support personnel, maintenance technicians, software, equipment, and fleets of vehicles. These providers are starting to look into preempting any potential failures if possible. Or, in the worst case, try to solve the problem from the first attempt, as soon as possible, with the least human interaction and, especially, client impact. Thus, call influx and duration in support centers can be reduced, money saved and client satisfaction is enhanced [1].

When it comes to maintenance solutions, we are interested in two kinds predictive-maintenance and prescriptive-maintenance. The difference is that predictive-maintenance is pre-occupied with predicting when faults may occur while prescriptive-maintenance builds on that and proposes solutions and actions that should be taken to correct the faults or even prevent them.

First, we looked at predictive-maintenance works [2]. The techniques used covered knowledge based approaches [3–7], traditional machine learning approaches [8–10] and deep learning approaches [11–14].

While predictive-maintenance techniques are widely documented in the literature, prescriptive-maintenance is still a very new discipline and very poorly documented. The methods used include probabilistic models [15], rule based [16] and machine learning models [17, 18]. The works mentioned in this review are representative and not comprehensive.

Even though the techniques employed varied, the entirety of these works rely only on sensor data in the form of digital measurement such as images, vibration signals, temperature measurements among others. However, in some cases as in the NOCs operations, data can be presented in the form of natural language. Indeed, even though part of that data is collected from sensors and machines on the network, a large part of that data is generated from the conversation between the clients and the support agent and recorded in natural language.

The work that we will describe in this paper is a knowledge based approach and more precisely an ontology based approach. It differs from other ontology based approaches [3, 19, 20] in that it uses semantics not only to capture the relations between the entities in the domain context knowledge but also to capture the information and establish relations from natural language data. It also differs in that it can use historical fault description to proposing solutions for new unseen fault description.

The work that we present in this paper describes the system we implemented toward a prescriptive-maintenance operation at Thales. The paper is organized as follows; First we describe the use case we worked on, the challenges that this use case presented and the objectives we worked toward achieving. Second, we describe the data used and the semantic representation that we designed to integrate this data into our solution. Then we describe the methods that we used to leverage the semantic representation of the data to create new useful data to prescribe solutions to old and new maintenance problems. Last, we conclude with our take-always from implementing such a solution in a NOC environment.

2 Use Case and Challenges

2.1 Thales Reflex Platform for Support Maintenance

The Thales Reflex Platform is ensuring data management services to telecommunication providers. Among this data, the platform aims at managing error codes logs from the customer devices: mainly internet, phones and TV services. As part of the services provided to clients, the platform exposed a new capability to better exploit historical and real-time data of problems and solutions, to allow technicians in NOCs to rapidly, and even before the fact, provide solutions to problems that are not in the customer support textbook but may have been successful in resolving problems by some in the past or recently (from historical data) [1]. As a first target, the Thales Reflex Platform's aim is to reduce the average call duration from an average of 7 min to 5 min and save 10% of operating cost that amounts to a 56 million dollars savings. This operation touches on tens of million eventual operator customers and about 27.5 million customers. The Reflex Platform main operation consists of collecting multimodal data from the different devices on the network, organizing this data in a semantic framework, using machine learning and a reasoning model to identify faults on this network and prescribing actions either to the devices themselves or to the customers to correct the fault. The platform also reports these faults and the prescribed solutions in the NOC (see Fig. 1.)

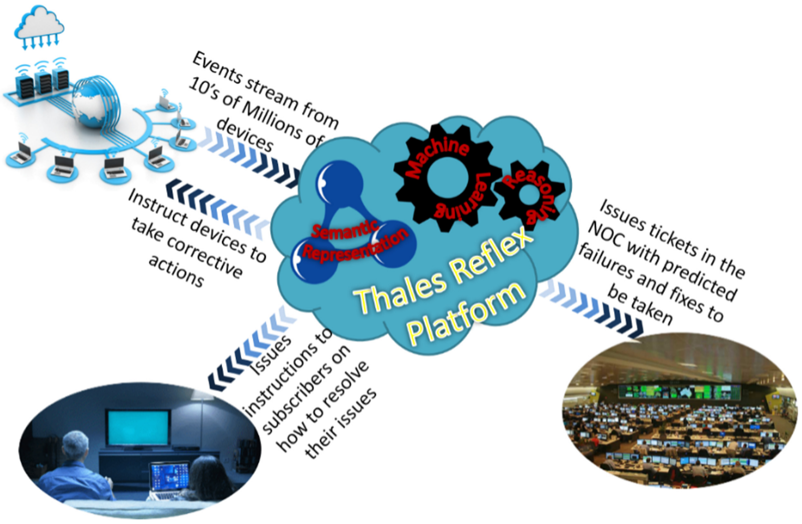
Events stream from 10's of Millions of devices
Instruct devices to take corrective actions
Semantic Representation
Machine Learning
Reasoning
Thales Reflex Platform
Issues tickets in the NOC with predicted failures and fixes to be taken
Issues instructions to subscribers on how to resolve their issues

Fig. 1. A schema showing the existing and targeted functionalities on the Thales Reflex Platform. Image adapted from [1] for illustration purposes.

2.2 Use Case

As a use case, we will present the work done to extend the Reflex platform. In this use case we exploited two types of data (textual and code) to put together a pipeline that prescribes resolutions to problems that happen on a telecom network. The problems can happen at the customer end or on the network itself. In a normal operation case, the customer (a subscriber to the service or another technician) calls the NOC to report a problem. This prompts the issue of a ticket. The ticket is a record of the problem details and the suggested solutions. The agent at the NOC, transcribes the information in a natural language format and suggests actions to the customer (ex: reboot the device) or takes actions to resolve the problem. More than one action can be needed to resolve the problem. These are also recorded in a natural language format in the ticket record. To drive this process efficiently, and reduce the call duration and the time it takes to propose the right solution, we looked at ways to augment the process with semantic representation of tickets and problems as well as machine learning to expose prescriptive actions.

2.3 Challenges

When looking at the available data and analyzing the requirement of the problem we realized that we are facing several challenges. First, the search space is extremely wide. The number of problems that can be faced on the platform is not trivial and the syntax to express these problems is also largely varied. The data is noisy. The problem description is seldom accurate and the syntax used to express these problems could also be vague. In addition, the solution proposed and documented by the support agent is not guaranteed to be the "right" solution. There is no indication if the problem was definitely solved during the interaction. We also realized that there is no standard taxonomy or ontology for this particular type of data. The latter is crucial to structure the information, the tickets, challenges and potential solutions which could be re-usable among similar semantic cases.

2.4 Objectives and Solution Motivation

Our ultimate objective is to build a prescriptive-maintenance system which will rely on semantic representation of information, and then exploit structure representation through a hybrid combination of machine learning and reasoning paradigms. The system will prescribe potential solutions when presented with certain symptoms or similarities with other cases. In particular, the engine should prescribe fixes when presented with the description of a problem on a digital telecommunication platform and other relevant information.

The solution is not diagnostic. We do not particularly aim at understanding the problem in itself or validate it when it is presented. We are more interested in proposing solutions when certain symptoms are presented such as a problem description (which could be accurate or not). The reasoning engine is examining historical relational data (the information collected in the available ticket reports as described in Sect. 3) and uses analogical reasoning (on top of semantically augmented data) to derive conclusions. The engine is not limited to proposing solutions only for problems that are described in historical data or some given guidelines. It is also required to proposing solutions for unseen problems.

Given that the data is multimodal, the solution encodes this data in a unified format that captures its semantics and the relationships between the entities of the context domain.

The number of faults that occur on networks covered by the Reflex Platform is extremely large. On a ticket, each of these faults is non-uniformly expressed given that they are recorded in natural language. This creates a large domain space and, in practice, the data available to represent this space will be relatively sparse. For this reason, the solution that we adopted is not enormously data hungry.

The reasoning engine pipeline that we implemented is shown in Fig. 2.

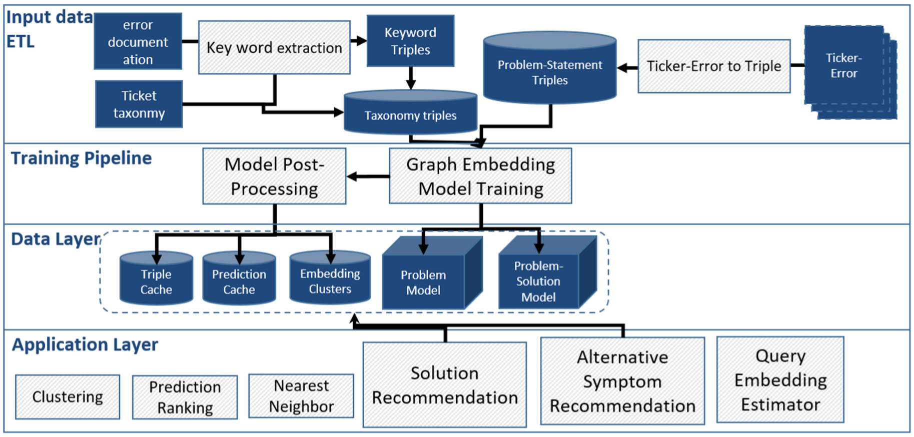

Fig. 2. The proposed reasoning engine pipeline. The blue blocks refer to data instances or repositories and the grey blocks are processes. The upper most layer describes the data preprocessing and structuring. The following layer describes the data parsing into vectors in the embedding space that encode semantic relational information. The third layer describes the different cache used to allow the pipeline to be used in real time. The most bottom layer describes the solution recommendation part of the reasoning engine process. (Color figure online)

3 Data

A three months period of ticket and error codes data was collected for model training purposes. This reaches a total of 3.6 million ticket records, and 61 million error code records, which amount to approximately 30 GB. We report the experiments on the ticket/error code data set.

The ticket dataset contains individual entries summarizing details of a problem reported to a support agent. The tickets, in this dataset, have been logged by support agents.

The error code dataset contains codes collected in the network. These error codes are generated at the device level. A problem can generate a collection of error codes, and also error codes that are not unique to one problem. The ideal solution would be to guess the combination list of error codes generated by a certain problem. However, the association between such a list and a problem is not straight forward. Also, the problem reports are not punctual. A problem is usually detected much later than its occurrence. It is the same for the error codes generated. Also devices frequently generate “benign” errors codes. Given that there is no indication of when the problem(s) happened or a way to distinguish “benign” from “problematic” codes, the association between the problem and its combination of generated error codes is far from straightforward. For the purposes of this project, we associated a problem to the list of error codes that occurred in the interval of the 7 days of which the reporting time is in the middle. This assumption is adopted by the technicians themselves. Of course, this highly inaccurate assumption creates a highly noisy association between the reported problems and the list of error codes associated with each one.

We also have a dictionary that associates each error code to potential problem descriptions.

We constructed a dataset of 600 k ticket-error code pairs by sampling over the original 61 M records. This resulted in 35 k reported tickets relating to two types of services (“hsd service” and “video service”). We limited our analysis to these services because they had enough coverage in the ticket dataset and are of the most interest for our predictive-maintenance platform. These ticket-error code pairs covered a set of device manufacturers and each ticket (problem) report is associated on average with 20 error codes.

4 Semantic Representation

Due to the specialized nature of our problem and dataset, it was crucial to have a dedicated ontology (see Fig. 3) to populate a knowledge graph of tickets, related by potential symptoms and solutions (see Fig. 4).

4.1 Domain Ontology

The domain ontology consisted mainly of ten major high-level classes:

Ticket: represents a certain ticket and each instance expressed using a unique ID (ex: DI0618002227)

Date/Time: indicating the date time that a ticket was reported.

Account: represents the customer account. Each instance is identified by a unique ID.

Error: represents the error code generated by the hardware and potentially assigned to a problem. (ex: b20f8ee03140218f)

Problem: describes the problem reported by a customer (ex: video quality problem)

Manufacturer: represents the manufacturer of a certain device potentially affected by a reported problem (ex: dwalin).

Device: represents a device that is potentially affected by a reported problem. (ex: 1:px022mine)

Resolution: represents the solution proposed by the agent. This is however not a guarantee appropriate solution. (ex: power cycled device)

ProblemDescription: This is the description provided by the ticket (ex: unable to play VOD assets)

Network: The network affected or that the device is branched to. (ex: eriador-minhriath)

The seven classes are related by the following relations *fromAccount*, *causedBy*, *causeType*, *affectsDevice*, *manufacturer*, *Network*. Using these relationships we formed 820 k triples after filtering the duplicate errors codes. Figure 3 shows a simplified version of the ontology graph.

When looking at the *Problem*, *Resolution* and *Error* classes, each had an enormous extended taxonomy (the tree of subclasses was very shallow) that was not exploitable in this case. Keeping up with this degree of detail was not advantageous. For this reason, we decided to create a certain hierarchy and group these subclasses into new mega-subclasses according to their semantic similarities. These mega-subclasses were related to their respective classes via the relationship (Problem, Resolution and Error) *isA* thus forming 20 k triples in total.

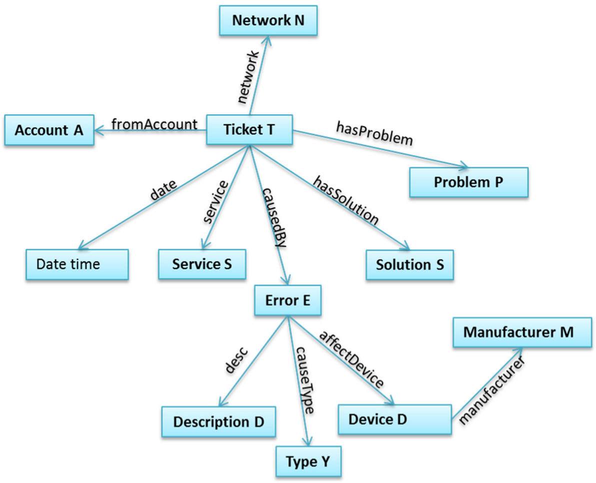

Fig. 3. A simplified version of the ontology (encoded in RDF).

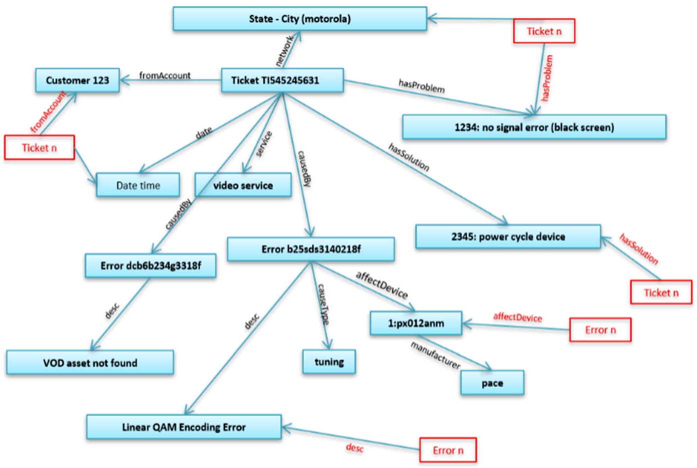

Fig. 4. A local snapshot of a neighborhood in the training data graph. Some values are fictitious for client privacy consideration.

4.2 Knowledge Graph

Once all data is structured following ontology in Fig. 3, the result is a knowledge graph that contains the entire relational information in the dataset. Figure 4 shows a snapshot of the knowledge graph. As our task is to offer solutions to, not only seen requests, but also new or similar ones using the knowledge that we collected and structured, it is important to structure the domain information in a format that could be linked with external vocabularies such as DBPedia [21]. This is crucial to link entities and potentially identify commonalities among them through external links. This ensures to obtain a domain knowledge graph which is contextualized but also interpretable across other domains.

By deriving a knowledge graph structure we can 1) encode explicitly domain knowledge through relations, 2) add external knowledge through external vocabularies, 3) design the problem of identifying solutions of a problem as the task of link prediction in a knowledge graph, 4) identify exact solution or partial solutions through semantic relatedness in the graph. The latter step is tackled through missing link prediction, combining machine learning on knowledge graphs.

5 Machine Learning on Knowledge Graphs for Link Prediction

To be able to exploit the relational information contained in the knowledge graph, and then derive potential solutions of new problems, we map the problem as a link prediction task in a knowledge graph. A problem P and solutions S are classes in the graph, and weighted links Rs between them are the likelihood of the solutions in S to solve P. We addressed this problem by exploiting machine learning on knowledge graphs through graph node embedding [22, 23].

The goal of a graph node embedding is to parse the nodes of a graph in a low-dimensional vector space such that the optimized vector space encodes the structure or relations of the original graph. In this space, also, the relationship, or the edges between the nodes are represented by geometric relationships. Vectors in the embedding space (to which we will simply refer to as embeddings in the rest of the paper) do not only encode information about the entity itself but also about its position and relationships in the graph.

We evaluated four embedding algorithms on the same dataset; TransE [24], DistMult [25], and HolE [26] and ComplEx [27]. We based our judgment on quantitative assessment of the result of each of the algorithms. Results of this particular analysis are not reported in the paper due to page limits[1]. ComplEx was retained for converging to better results during training. These results are consistent with reported results in the literature [23, 27]. The ComplEx algorithm represents each node and the links *Rs* that connect them with complex-valued embeddings. The algorithm then calculates the score for each triple (*head*, *tail*, *relation*) as the real part of the trilinear product of the corresponding embeddings. Overall, the algorithm optimizes the values of the embeddings so that the

[1] The details and results of the embedding algorithms comparison are at http://www-sop.inria.fr/members/Freddy.Lecue/thales/iswc-2020-in-use-PrescriptiveMaintenance-extra-results.pdf.

nodes that are semantically related are attributed a high score while the nodes that are not related are attributed low scores.

We used the implementation of ComplEx as is in the platform Ampligraph [28].

6 Solution Reconstruction for Support Maintenance

In this context, given a problem, predicting its solution would have been to add a corresponding node in the graph and predicting a link to a “best” solution node in the graph. However, we did not adopt this method for two reasons. 1) Generating embedding space for new nodes in a graph is not a straightforward task. Adding a new node to the graph will affect its neighborhood in the least and will relatively affect the embedding values of all the nodes of the graph. Repeating the complete space embedding calculation process for each new query is not a practical solution. It is computationally exhaustive and it will not suit real time applications. 2) We wanted to emulate the reasoning process of the support technician. We wanted to look at the error code space and attempt to find patterns or deviations from those patterns that can be leveraged not only to propose a one shot solution but to be able to create a reasoning pipeline that is able to propose a solution and alternative solutions that are not similar but complimentary. In case the first solution failed, we wanted to propose other options that are independent and not similar and that can solve that same problem. That is why the knowledge graph structure is crucial in our settings. We suspected that the best scored solutions would be similar and propose similar approaches.

6.1 Vector Approximation in the Embedding Space

We evaluate the embedding vector of a new problem as an approximation of semantically close problem embeddings. To this end we calculate the embedding of a new problem node as the mean embedding of problem nodes having similar linked error codes. The rationale is that similar problems produce similar error codes. A problem generated on a given network, related to given devices by given manufacturers, manifesting in a given manner and so on, also manifest a similar set of error codes every time. Following this logic, problems with a similar list of error codes have similar attributes and, as a result, similar embeddings.

```
$errors_{PN}$ = the list of errors associated with $problemN$
for $error_n$ in $errors_{PN}$
    $problems_{en}$ = list of problems associated with $error_n$
    for $problem_y$ in $problems_{en}$
        $errors_y$ = the list of errors associated with $problem_y$
        $score_y = |errors_{PN} \cap errors_y| / |errors_y|$
$embedding_{ProblemN}$ = average of the five problem embeddings with highest scores
```

Fig. 5. A pseudo code that describes the procedure to approximate new problem embeddings.

Also, in the context of this project a new problem, *problemN* (represented by an embedding, $embedding_N$), does not create new attributes (i.e., network, device, customer, error codes, etc.). A new problem has a new ID and a new combination of individual attributes. Those individual attributes are already represented by nodes in the graph and have their already computed embeddings. Then, for each of the error codes in the error code list attributed to *problemN*, we calculate the problems, in the graph, that are linked to that error code. We score the problems by the number of error codes that they share with *problemN* and average the embeddings of the best five scored problems to approximate the $embedding_N$. This procedure is described in the pseudo code of Fig. 5.

6.2 Solution Recommendation

Our aim is to emulate the process of problem troubleshooting that an agent has during an interaction with a customer facing a problem. We proposed a process formed of a set of condition-based instructions that aims at maximizing the confidence in each consequent proposed solution. Our stepwise approach (see Fig. 6) is as follows:

Step1. Verify if the exact problem exists in our dataset. We do so by calculating the Euclidean distance between $embedding_N$ and the other problem embeddings. If the best distance is lower than a threshold, **propose the solution associated with the problem that corresponds to that distance.**
Step 2. Use K-means [29, 30] and pre-cluster all the problem embeddings in the dataset into K clusters using the Euclidean Distance.
Step 3. Calculate the *Nearest Neighbors (NN)* problem embeddings of the *problemN* embedding. Then, group these embeddings according to the clusters calculated in step 1 (see Fig. 5). Our justification for clustering the problems is that we considered the shared solutions of problems that are similar to *problemN* to constitute the recommended list of solutions to *problemN*. Also we considered multiple problems instead of just one (the nearest one) to minimize the impact of accumulated error codes throughout the complete pipeline.
When computing the neighbors, we considered three metrics to calculate the distance between the embeddings: the Euclidean distance, the cosine similarity distance and the hyperbolic distance. The Euclidean distance and the cosine distance returned almost similar results. However, we adopted the hyperbolic distance because it better emphasized the relevant differences between the problems (ex: one additional error code between two problems with the same list or error codes).
Step 4. Use ComplEx to perform link prediction between the *NN* problem nodes and solution nodes, and recommend solutions for each of the *NN* problems (see Fig. 6). Each of the solutions is predicted with a confidence score.

Step 4.1. If all the *NN* problems agree on one solution, **then propose this solution.**
Step 4.2. If all the *NN* problems fall under one cluster ($cluster_x$), then we consider that the proposed solution is the synthesis of the total of the solutions proposed by this cluster (*solution_nnList*). The solution embedding ($embedding_S$) to *problemN* is calculated as such

$$embedding_S = embedding_N - weightedAvg(cluster_x) + avg(solution\_nnList) \quad (1)$$

The average is weighted by the confidence associated with each predicted solution by ComplEx. **Propose the solution with the closest embedding to *embedding_solution*.**

Step 4.3. If the conditions in Step 4.1 and Step 4.2 are not met, our confidence in the one possible solution is decreased and we aim at proposing a list of two possible solutions as follows

Step 4.3.1. Calculate the average embedding of each group of problem embeddings calculated in Step 2
Step 4.3.2. Maximize the following distance for each combination possible

$$Distance_{S_M^{GA}, S_N^{GB}} = argmax_{S_M^{GA}, S_N^{GB}}\left(|GA - GB| + \left|S_M^{GA} - S_N^{GB}\right|\right) \qquad (2)$$

where GA and GB are the two average embeddings associated with each of the two groups, respectively. Also, S_M^{GA} is one of the solutions associated to one of the problems in one of the problem groups GA and S_N^{GB} is one of the solutions associated with one of the problems in another of the problem groups GB. Where $S_M^{GA} - S_N^{GB}$ is the distance between S_M^{GA} and S_N^{GB}.

Propose the List of Two Solutions S_M^{GA} And S_N^{GB}
In choosing such a combination, we aim to ensure that the solutions we are proposing are alternative solutions and not similar solutions.

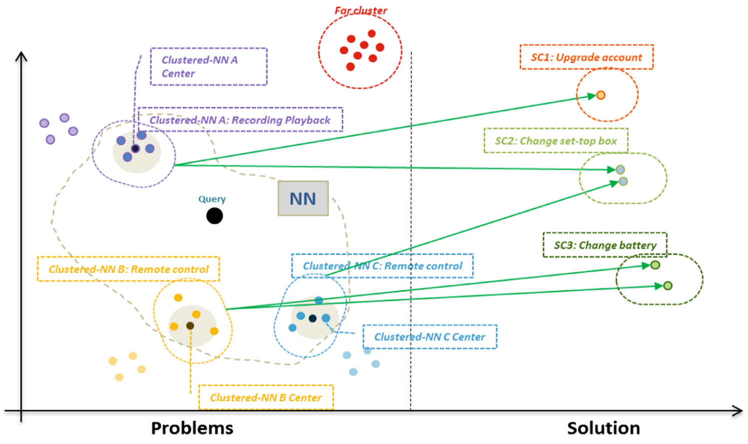

Fig. 6. The problems are grouped according to the clusters that highlight their semantic similarity. Also, each group has its mean embedding calculated. We used ComplEx to estimate the solutions (right plane) associated to each *Nearest Neighbors(NN) problems* (left plane)

Experiment and Results. We tested the proposed solution recommendation algorithm on a dataset that included problem samples spanning 76 solution classes. The dataset also consisted of 5,000 problems and was unbalanced with respect to the solution classes (realistic case). We considered a hit when the recommender returned the real solution in a list of 5. It is important to keep in mind that when we query the ComplEx algorithm to establish a link (make an association), the algorithm calculates the probability of a link between the given entity and others in the graph. For these results we considered the 5 most probable links to solution nodes. The results are summarized in Table 1.

Lessons Learnt. The results achieved in this work reach the level of readiness for integration in the platform for deployment and in-use in the context of the telecommunication scenario. The limitation factors of the approach are: (1) the high uncertainty in the relations established in the training dataset, in particular the association between the list of

```
Let $embedding_N$ be the embedding that represents problemN in the embed-
ding space
Let embeddingList be the list of embeddings in the embedding space that
belong to the problems on the database
$Embedding_y \in embeddingList$
$Distance_{min}$ = argmin_y({Euclidean_Distance($embedding_y$, $embedding_N$})

Let $embedding_{min}$ = the embedding in embeddingList that is associated with
$distance_{min}$
Let $solution_{min}$ = the solution represented by $embedding_{min}$
if $distance_{min} < threshold_n$
  propose the solution represented by $solution_{min}$
else
    $cluster_{list}$ = k_means(embeddingList)
    nn_neighbourList = nearest_neighbours($embedding_N$, embeddingList)

    for $neighbour_{nn}$ in solution_nnList
      $solution_{nn}$ = ComplEx($neighbour_{nn}$)

    Let solution_nnList = {$solution_{nn}$| $solution_{nn}$ is unique}
    Let $cluster_x \in clusterList$
    if solution_nnList contains one element solution embedding
      propose the solution represented by that solution embedding

    elif $solution\_nnList \subset cluster_x$
         $embedding_S$ = ($embedding_N$ - weighted_avg($cluster_x$)) +
         avg(solution_nnList)
         propose the solution represented by the embedding that is clos-
         est to $embedding_S$
    else
        Let $group_g$ = {$embedding_x \in cluster_x$ and $neighbour_{nn}$ |$cluster_x$
         clusterList, $neighbour_{nn} \in solution\_nnList$ }
      Let G = {avg($group_g$)}
      Let $GA \in G$ and $GB \in G$
         Let $S_M^{GA} \in solution\_nnList$ and $group_A \in group_g$ | $S_M^{GA}$ is associat-
         ed with one of the problems in $group_A$
         Let $S_N^{GB} \in solution\_nnList$ and $group_B \in group_g$ | $S_N^{GB}$ is associat-
         ed with one of the problems in $group_B$
      $Distance_{S_M^{GA},S_N^{GB}}$=$argmax_{GA,GB,M,N}$({|GA - GB| + |$S_M^{GA}$ - $S_N^{GB}$|})
      propose the two solutions that corresponds to $S_M^{GA}$ and $S_N^{GB}$
```

Fig. 7. Pseudocode describing the solution recommendation process

Table 1. The solution recommendation results.

Accuracy	Precision	Recall	F1 score
0.58	0.69	0.58	0.60

error codes and the problem and the relation between the problem and the solution, (2) initial inaccuracy in problems reporting.

6.3 Alternative Solution Recommendation

Usually the problem description in the ticket is not accurate, or the list of associated error codes is missing a key error code that should be associated with the given problem. This results in a solution recommendation with low confidence that manifests as low scores in the solution proposal in 3.1, 3.2 or 3.3.4. Normally, the agent has to ask a long series of questions to resolve the problem. The situation usually annoys the customer and costs a lot of money in time spent over the phone or necessitating the dispatching of a technician on location. We address this issue and theorize that similar problems are represented by similar embeddings and also have similar error code lists.

Our method consists of proposing a list of *P* (in this case 5) solutions as follows:

Step 1. Among the nearest neighbors of *problemN*, consider the one with the highest score associated to its solution (the correlation between the solution and the problem is high)
Step 2. Look at the error codes associated with the neighbor and not associated with *problemN*. We assume that these are the error codes that were missing from the original list of error codes associated with *problemN*.
Step 3. Add the solutions associated with these error codes to the list of alternative recommendation solutions.
Step 4. Relax the threshold on the nearest neighbors calculation (widen the neighborhood) and reiterate steps 1, 2 and 3 until the number *P* of items in the list is met.

Experiment and Results. To test the alternative solutions recommendation procedure we had to build a special dataset for evaluation purpose. We needed problems that are replicas of existing problems (having the same problem description and having the same list of error codes associated with them) except for one error code missing from the list associated with one problem and existing in the other. These cases were hard to find in our dataset so we built such problems by randomly selecting 15 thousand problems and cloning them and their associated list of error codes. For each of the clones, we removed one error code from the list. We made sure that no replicas of the clones exist in the dataset or if a clone exists the solutions associated with it are different from the solutions associated with the original problem (parent problem of the clone). We considered a hit when the right solution was among the list of 5. See the results in Table 2.

Table 2. The results for the alternative solution recommendation method

Accuracy	Precision	Recall	F1 score
0.82	0.84	0.82	0.83

These results are very encouraging. If a given query (*problemN*) has a low solution prediction score, the alternative solution recommendation algorithm is of interest. Considering other related non-occurring errors codes would help the technician finding more related problems and further investigating more plausible solutions.

Lessons Learnt. When looking at these results we can conclude that the algorithm achieved high accuracy in encoding the semantic relation between a given list of error codes and its corresponding problem. These results were even robust with respect to incomplete or noisy (missing or extra codes) lists.

6.4 Automated Chatbot Support Agent

A chatbot was developed for a customer support agent to exploit the platform and quickly understand/differentiate the caller's problem and recommend solutions. The chatbot uses the customer's problem description to initiate the interaction. Through an iterative process of question and answers, the chatbot refines its understanding of the current problem and begins suggesting solutions when it reaches the appropriate level of certainty in the problem and applicable solution. A video[2] is available to demonstrate the functionalities of the chatbot.

7 Conclusion

We presented work we deployed to reduce phone call durations in NOC and to increase customer satisfaction. We implemented and deployed a reasoning engine pipeline that, based on structured historical data, was able to establish connections between new unseen problems and historical documented problems and propose a list of potential solutions. All information has been encoded in a knowledge graph for encoding its semantics, and then we expose the problem as a link prediction problem in a knowledge graph. Even though the data captured uncertainty the method reached level ready for deployment in our platform and testing with a client for support maintenance. We identified several exploration venues to improve further the results. First, we need to address the issue of identifying the list of malign error codes associated with a certain problem. We are aware that this list will never be deterministic or accurate otherwise the problem of finding the solution will become straight forward. However efforts should be made to

[2] http://www-sop.inria.fr/members/Freddy.Lecue/thales/iswc-2020-in-use-PrescriptiveMaintenance.mp4.

lower the uncertainty in selecting the error codes. Second, we are also aware that the quality of the embedding calculated for new entities is not sufficient. Efforts should be made to construct new knowledge graph embeddings for new entities without having to execute the complete embedding generation algorithm. This is still an emerging research question and we plan to address it in our future work.

Finally, the work described in this paper consists of a very valuable feature of the Reflex platform. It also gives the platform an edge in the domain of prescriptive-maintenance and customer support.

Acknowledgements. The authors would like to thank Dr. Roger Brooks for his support along the duration of the project.

References

1. Watson, J., Brooks, R., Colby, A., Kumar, P., Malhotra, A., Jain, M.: Predicting service impairments from set-top box errors in near real-time and what to do about It. In: the 2018 Fall Technical Forum (2018)
2. Ran, Y., Zhou, X., Lin, P., Wen, Y., Deng, R.: A survey of predictive maintenance: systems, purposes and approaches. arXiv:1912.07383 (2019)
3. Konys, A.: An ontology-based knowledge modelling for a sustainability assessment domain. Sustainability **10**(2), 300 (2018)
4. Peng, Y., Dong, M., Zuo, M.J.: Current status of machine prognostics in condition-based maintenance: a review. Int. J. Adv. Manuf. Technol. **50**(1), 297–313 (2010)
5. Zhang, Z., Si, X., Hu, C., Lei, Y.: Degradation data analysis and remaining useful life estimation: a review on Wiener-process-based methods. Eur. J. Oper. Res. **271**(3), 775–796 (2018)
6. Olde Keizer, M.C.A., Flapper, S.D.P., Teunter, R.H.: Condition-based maintenance policies for systems with multiple dependent components: a review. Eur. J. Oper. Res. **261**(2), 405–420 (2017)
7. Hong, H.P., Zhou, W., Zhang, S., Ye, W.: Optimal condition-based maintenance decisions for systems with dependent stochastic degradation of components. Reliab. Eng. Syst. Saf. **121**, 276–288 (2014)
8. Rao, B.K.N., Pai, P.S., Nagabhushana, T.N.: Failure diagnosis and prognosis of rolling - element bearings using artificial neural networks: a critical overview. J. Phys: Conf. Ser. **364**, 012023 (2012)
9. Susto, G.A., Schirru, A., Pampuri, S., McLoone, S., Beghi, A.: Machine learning for predictive maintenance: a multiple classifier approach. IEEE Trans. Ind. Inform. **11**(3), 812–820 (2015)
10. Chen, X., Wang, P., Hao, Y., Zhao, M.: Evidential KNN-based condition monitoring and early warning method with applications in power plant. Neurocomputing **315**, 18–32 (2018)
11. Guo, L., Lei, Y., Li, N., Yan, T., Li, N.: Machinery health indicator construction based on convolutional neural networks considering trend burr. Neurocomputing **292**, 142–150 (2018)
12. Zhu, J., Chen, N., Peng, W.: Estimation of bearing remaining useful life based on multiscale convolutional neural network. IEEE Trans. Ind. Electron. **66**(4), 3208–3216 (2019)
13. Wu, Y., Yuan, M., Dong, S., Lin, L., Liu, Y.: Remaining useful life estimation of engineered systems using vanilla LSTM neural networks. Neurocomputing **275**, 167–179 (2018)
14. Deutsch, J., He, D.: Using deep learning-based approach to predict remaining useful life of rotating components. IEEE Trans. Syst. Man Cybern.: Syst. **48**(1), 11–20 (2018)

15. Ansari, F., Glawar, R., Sihn, W.: Prescriptive maintenance of CPPS by integrating multimodal data with dynamic bayesian networks. Machine Learning for Cyber Physical Systems. TA, vol. 11, pp. 1–8. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-662-59084-3_1
16. Matyas, K., Nemeth, T., Kovacs, K., Glawar, R.: A procedural approach for realizing prescriptive maintenance planning in manufacturing industries. CIRP Ann. – Manuf. Technol. **66**, 461–464 (2017)
17. Goyal, A., et al.: Asset health management using predictive and prescriptive analytics for the electric power grid. IBM J. Res. Devel. **60**(1), 4:1–4:14 (2016)
18. Ansari, F., Glawar, R., Nemeh, T.: PriMa: a prescriptive maintenance model for cyber-physical production systems. Int. J. Comput. Integr. Manuf. **32**(4:5), 482–503 (2019)
19. Schmidt, B., Wang, L., Galar, D.: Semantic framework for predictive maintenance in a cloud environment. In: Proceedings of the 10th CIRP, vol. 62, pp. 583–588 (2017)
20. Medina-Oliva, G., Voisin, A., Monnin, M., Leger, J.-B.: Predictive diagnosis based on a fleet-wide ontology approach. Knowl.-Based Syst. **68**, 40–57 (2014)
21. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: a nucleus for a web of open data. In: Aberer, K., et al. (eds.) ASWC/ISWC -2007. LNCS, vol. 4825, pp. 722–735. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76298-0_52
22. Zhang, S., Tay, Y., Yao, L., Liu, Q.: Quaternion knowledge graph embeddings. In: NIPS, pp. 2735–2745 (2019)
23. Wang, Q., Mao, Z., Wang, B., Guo, L.: Knowledge graph embedding: a survey of approaches and applications. IEEE Trans. Knowl. Data Eng. **29**(12), 2724–2743 (2017)
24. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: NIPS, pp. 2787–2795 (2013)
25. Yang, B., Yih, W., He, X., Gao, J., Deng, L.: Embedding entities and relations for learning and inference in knowledge bases. In: 3rd International Conference on Learning Representations, (ICLR 2015) San Diego (2015)
26. Nickel, M., Rosasco, L., Poggio, T.: Holographic embeddings of knowledge graphs. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, pp. 1955–1961, Phoenix, Arizona (2016)
27. Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., Bouchard, G.: Complex embeddings for simple link prediction. In: Proceedings of the 33rd International Conference on International Conference on Machine Learning, vol. 48, pp. 2071–2080. New York (2016)
28. Costabello, L., Pai, S., Van, C. L., McGrath, R., McCarthy, N.: AmpliGraph: a Library for Representation Learning on Knowledge Graphs, Zenodo (2019)
29. Forgy, E.W.: Cluster analysis of multivariate data: efficiency versus interpretability of classifications. Biometrics **21**(3), 768–769 (1965)
30. Lloyd, S.: Least squares quantization in PCM. IEEE Trans. Inf. Theory **28**(2), 129–137 (1982)