Towards Understanding and Analyzing Rationale in Commit Messages using a Knowledge Graph Approach

Mouna Dhaouadi
DIRO
Université de Montréal
Montréal, Canada
mouna.dhaouadi@umontreal.ca

Bentley James Oakes
DIRO, Université de Montréal
GIGL, Polytechnique Montréal
Montréal, Canada
bentley.oakes@polymtl.ca

Michalis Famelis
DIRO
Université de Montréal
Montréal, Canada
famelis@iro.umontreal.ca

***Abstract*—Extracting rationale information from commit messages allows developers to better understand a system and its past development. Here we present our ongoing work on the Kantara end-to-end rationale reconstruction pipeline to a) structure rationale information in an ontologically-based knowledge graph, b) extract and classify this information from commits, and c) produce analysis reports and visualizations for developers. We also present our work on creating a labelled dataset for our running example of the Out-of-Memory component of the Linux kernel. This dataset is used as ground truth for our evaluation of NLP classification techniques which show promising results, especially the multi-classification technique XGBoost.**

***Index Terms*—rationale structuring, rationale extraction, Natural Language Processing, Linux, ontology, dataset, openCAESAR**

I. Introduction

The development of software systems involves constant decision-making at various levels of abstraction and degrees of importance. For every such decision, the system developer has a *rationale*, i.e., the *why* reasoning that explains their decision. Rationale may be more or less explicitly articulated and is often left implicit or unrecorded. Regardless, it is a very useful piece of information for the development team, as it can be used to learn from mistakes or reuse solution patterns [1]. Many researchers have thus attempted to extract rationale and to help developers leverage it [2]–[5].

In this article, we focus on the rationale expressed in developer commit messages submitted to version control systems such as Git. Tian *et al.* have found that a quality of a "good" commit message is that it contains rationale information [6]. Such information can help other developers understand and contextualize the proposed changes with respect to other project commits. Crucially, it also allows other stakeholders to assess the *quality* of the proposed changes and how well the changes fit with the project requirements. Over time, these messages help build a shared understanding of the design and behavior of the software system among the development team.

The primary challenge to understanding rationale is that rationale information in commit messages is not typically expressed in a structured representation or model, and instead is embedded in natural language text. Therefore, identifying and understanding the rationale depends on deep prior knowledge of the project context. This increases the mental effort required to understand a commit, its key decisions, and their rationale. From the point of view of maintaining a shared understanding, this makes it *harder to on-board new contributors*, and makes it *difficult to assess the quality of the provided rationale of each new commit*. Finally, it makes it difficult to *establish and maintain traceability* with decisions and rationale information elsewhere in the project.

Here, we here present our results towards extracting models of rationale from commit messages using the Kantara framework for end-to-end rationale reconstruction [7]. Kantara defines an *information inference component* to structure and extract rationale information, and an *analysis interface* to understand the extracted rationale. We apply a proof-of-concept demonstration of Kantara to a set of 180 commits from a Linux kernel subsystem. The main contribution is showcasing the practicality of end-to-end rationale extraction and analysis for a real world software project, and highlighting the challenges.

In Section II we explain our running example of the OOM-Killer subsystem in the Linux kernal, our approach to labelling this subsystem's commits with the rationale information, and discuss our insights on how kernel developers explain their commits. The extraction of rationale information from commits is presented in Section III, where Natural Language Processing (NLP) techniques are used to automatically classify commit sentences. From our ground truth dataset, we quantitatively evaluate the performance of our classifiers. Section IV provides our contribution on the structuring of this rationale information in an ontologically-based *knowledge graph* [8], along with our efforts on representing this graph as an ontology in the openCAESAR framework [9] to leverage ontological semantics for inferencing. Section V discusses the analysis of the rationale information once it is placed in the ontological format. We present a query operating on this graph to extract valuable information about commit sentences, and an interactive visualization allowing a user to better understand how rationale is presented in commits by different authors. In Section VI, we overview related work, while Section VII concludes with a discussion on challenges we faced and the future work to address them.

II. Running Example: Linux Subsystem

This section will discuss our running example of the *Out-of-Memory* (OOM) memory management subsystem in the Linux kernel. First, the kernel itself and relevant development information is presented. Then, we discuss the OOM subsystem itself. Finally, we discuss our ongoing work on labelling rationale information in the commits of the OOM subsystem.

A. Linux Kernel and Development

Since its creation by Linus Torvalds in 1991, the Linux kernel has grown to run on devices ranging from mobile phones and tablets to supercomputers. This wide-spread adoption is partially due to its open-source and collaborative development.

The main development channel for Linux is the Linux Kernel Mailing List (LKML)[1]. The LKML contains email threads concerning every aspect of Linux development, including bug reports and potential patches. For example, this email[2] is a request for comment (RFC) on a potential patch to be applied to the kernel. Once the patches are accepted by a *subsystem maintainer*, they are passed through the maintainer hierarchy until the patches arrive at Linus Torvalds himself to be applied and released as a new version of Linux.

To maintain a high degree of quality in the Linux kernel, there are requirements for patch submission[3]. Among these, the subject line for the email containing the patch must fit a particular standard. This patch email must also contain a few lines or paragraphs describing the motivation/rationale behind the patch, and the impact on the kernel the patch will have. Traceability information is provided in multiple ways, such as a) patches are encouraged to add explicit links to LKML discussions in the patch descriptions, and b) patches must have a *summary phrase* (such as "oom: give the dying task a higher priority") which allows for tracing this patch across commits and LKML discussions. Linux kernel commit messages are thus a comprehensive repository of decision/rationale information to evaluate our approach.

B. Out-of-Memory Killer (OOM-Killer) Subsystem

The OOM-Killer kernel subsystem frees up memory when tasks have requested all available memory, preventing the system from crashing due to the lack of available memory. The OOM-Killer performs two primary tasks: a) it selects a task to kill, then b) it forces that task (the *OOM victim*) to release its memory and exit. The selection process is particularly relevant for our examination, as there are rich discussions about the best way to select the task to kill. For example, one code change was to reduce the chance of selecting kernel tasks specifically, later reversed in favor of a uniform selection process.

[1]Various archives exist for the LKML such as https://lkml.org/.

[2]https://lore.kernel.org/all/20100527180431.GP13035@uudg.org/

[3]https://www.kernel.org/doc/html/latest/process/submitting-patches.html.

TABLE I: Codebook

Label	Meaning
Decision	An action or a change that has been made, including a description of the patch behaviour
Rationale	Reason for a decision or value judgment
Supporting Facts	A narration of facts used to support a decision
Inapplicable	Pre-processing error or bad sentences (i.e., does not contain English sentences)

This OOM approach has been controversial since its origin in 1998, as some developers reason that the system should be allowed to crash in the presence of faulty software, or express concern that desirable user processes (such as a windowing environment) may be selected for killing.

C. Labelling Dataset for OOM Commits

To evaluate our Kantara framework (Section III), we require a form of ground truth to measure our extraction approaches. For this purpose, we are systematically creating a dataset of labelled commit messages for the OOM subsystem indicating the rationale information present in each commit sentence [10].

1) Commit Pre-processing: To create our data set, we selected 180 commits from the commit history of the OOM-killer file[4]. We did not include any merge commits, i.e., commits whose messages start with *Merge tag*. We pre-processed the messages of these commits: For each message, we removed the meta-data at the end of the message (e.g, information like *Signed-off-by* and *Suggested-by* were removed). We also removed URLs, references to other resources, and call traces using regular expressions. Afterwards, we split the message into sentences and kept only sentences with more than 3 characters and that are not source code. We identified the source code in the message body using heuristics like identifying keywords or symbols such as *git*, *$cd* or *$echo*. These keywords came from manually investigating the data.

2) Sentence Labelling Procedure: We performed six iterations of piloting rounds and consolidation meetings during which the three annotators (a PhD student, a post-doctoral researcher, and a professor) considered 38 randomly-chosen commits in total (which they annotated independently). Finally, we reached a consensus regarding the set of labels to use and our understanding of each label as shown in Table I.

We conducted the labelling by batches where if two annotators said *yes* to a label, then this was taken as the consensus. During the labelling process, Fleiss Kappa averaged 0.69 for eight rounds (so far). This indicates strong agreement considering the subjective nature of rationale [1].

3) Sentence Classification: Our rounds of labelling and consensus-building led us to further insights on how to classify each sentence. Consequently, we have updated our representation of rationale from what we had previously reported

[4]https://github.com/torvalds/linux/commits/master/mm/oom_kill.c accessed on 2023-01-12

Sentence	Labelling
mm, oom: base root bonus on current usage	Decision
A 3% of system memory bonus is sometimes too excessive in comparison to other processes.	Supporting Facts, Rationale
With commit a63d83f427fb ("oom: badness heuristic rewrite"), the OOM killer tries to avoid killing privileged tasks by subtracting 3% of overall memory (system or cgroup) from their per-task consumption.	Supporting Facts
But as a result, all root tasks that consume less than 3% of overall memory are considered equal, and so it only takes 33+ privileged tasks pushing the system out of memory for the OOM killer to do something stupid and kill dhclient or other root-owned processes.	Supporting Facts, Rationale
For example, on a 32G machine it can't tell the difference between the 1M agetty and the 10G fork bomb member.	Supporting Facts
The changelog describes this 3% boost as the equivalent to the global overcommit limit being 3% higher for privileged tasks, but this is not the same as discounting 3% of overall memory from *every privileged task individually* during OOM selection.	Supporting Facts
Replace the 3% of system memory bonus with a 3% of current memory usage bonus.	Decision
By giving root tasks a bonus that is proportional to their actual size, they remain comparable even when relatively small.	Rationale, Decision
In the example above, the OOM killer will discount the 1M agetty's 256 badness points down to 179, and the 10G fork bomb's 262144 points down to 183500 points and make the right choice, instead of discounting both to 0 and killing agetty because it's first in the task list.	Rationale

TABLE II: An example commit with labelled sentences from our dataset

in our earlier work [7]. In particular, as we now examine commits at a *sentence-level*, each sentence is given multiple classifications. We have also introduced the concept of a *supporting fact* to indicate sentences which do not themselves contain rationale, but instead present the current state of the system as explanatory text for the commit.

That is, a supporting fact is information in a sentence where a developer discusses the currently existing state of the system, at the moment before they propose a change. An example would be the description of the behaviour of a previous commit, such as the third sentence in Table II. Due to this reference to the past or current state of the system, these supporting facts are often found in the first half of the commit, and before the sentences containing rationale. In contrast, a *decision* provides information about the state of the system *after* the patch is applied, and thus it refers to the system's *future* state. *Rationale* is the *reason* for *why* a decision is taken, such as a *value judgement* about undesirable behavior.

Table II reproduces a commit from our dataset, along with a color-coded multi-label classification for each sentence. As an example of the labelling, the first sentence (the *summary phrase* of the commit) is labelled as a *decision* as it states the patch's change. The fourth sentence contains *rationale* as a value judgement ("something stupid") and *supporting facts* ("only takes 33+ privileged tasks").

4) Dataset Insights: An insight from this dataset is that there seems to be a trend for about 40-50% of the sentences in a commit message to contain rationale information [10]. For example, four out of nine sentences in Table II contain rationale. This trend could indicate a "natural" amount of rationale that developers include in their commit messages, or maybe a guideline for developers to target.

Figure 1 shows the distribution of the sentences over the identified categories[5]. This distribution motivates our creation of the *supporting facts* category, as there are many sentences that contain supporting facts but do not contain rationale. Also evident is the clear separation between sentences containing

[5]Note that the colouring shown in Figure 1 for each category is also reflected throughout other tables and figures in this article.

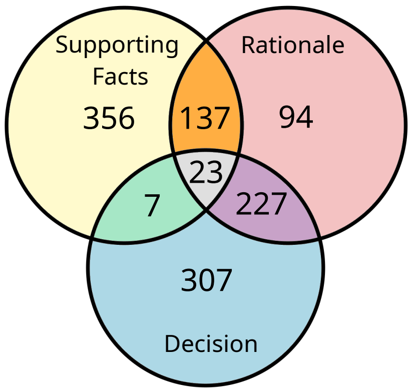


Fig. 1: Distribution of the sentences in our OOM dataset

decisions and those with *supporting facts*. However, the category of *rationale* is less clear-cut, with a substantial overlap between sentences with rationale and the other categories. One interpretation is that this indicates the subjective nature of rationale. Another possibility is that *rationale* is present in these sentences to motivate *decisions* and to provide value judgments on the existing state of the system as discussed in *supporting facts*. Examples of these multi-label sentences are found in Table II.

III. Extracting Rationale Information

This section will discuss a key part of the Kantara framework: the classification of sentences with respect to whether they contain *rationale*, *decisions*, or *supporting facts*. We discuss our approaches and provide an evaluation of our approach on our Linux subsystem dataset discussed in Section II-C.

Considering the subjective nature of the rationale information and the notable overlap between the different categories (Fig. 1), we investigated two classification approaches: *binary classification* (we do not consider the overlap), and *multi-label classification* (we consider the overlap). We selected the TF-IDF vectorizer to embed the commit messages. The vectors were then input into various ML models. We used the sklearn[6]

[6]https://scikit-learn.org/stable/index.html

TABLE III: Binary classification evaluation

Model	Decision				Rationale				Supporting Facts			
	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
Logistic Regression	0.72	0.75	0.20	0.30	**0.69**	0.20	0.03	0.05	0.70	**0.75**	0.13	0.21
Decision Tree	**0.78**	0.71	**0.61**	**0.64**	0.68	**0.51**	**0.57**	**0.53**	**0.71**	0.59	**0.46**	**0.51**
SVM	0.73	**0.85**	0.22	0.33	**0.69**	0.20	0.03	0.04	**0.71**	**0.75**	0.17	0.27

library for the models implementation and kept the default parameters values. The models were trained and evaluated using 10-fold cross-validation. That is, the data was randomly divided into ten equal splits, and nine of them were used for training and one for evaluating performance. We report the mean scores from these evaluations.

A. Binary classification

In binary classification problems, any of the samples from the dataset takes only one label out of two classes. We therefore consider the subset of the OOM dataset with the sentences that were labelled with only one label. Specifically, we ended up with 307 decision sentences, 94 rationale sentences and 356 supporting facts sentences. Since the data is imbalanced, we apply under-sampling to the majority classes and consider only 100 decision sentences and 100 rationale sentences. We then trained different classifiers considering one label at a time (i.e, when trying to classify decision sentences, rationale and supporting facts sentences we labelled as negative).

Table III reports the classification results of the widely-used binary classification models: Logistic Regression, Decision Tree and Support Vector Machines (SVM) [11]. We report the Accuracy, Precision, Recall and F1-score evaluation metrics.

From Table III, we extract four insights. First, the overall recall across the binary classification techniques is low. Second, the Decision Tree algorithm gave the overall best results. Third, the classification of the *Decision* sentences was more successful than the classification of the *Supporting Facts* sentences. Finally, the performance of the three classifiers when classifying *Rationale* sentences was rather low.

B. Multi-label classification

The multi-label classification is the most natural approach that applies to our problem as any sample from the dataset can be labelled with more than one label. In this experiment, we considered 1151 sentences (distributed as shown in Fig. 1) and we tried widely-used models for multi-label classification including the eXtreme Gradient Boosting (XGBoost) [12]. We report the micro-averaged evaluation results over the three categories in Table IV. Results indicate that the XGBoost classifier gave the overall best performance.

We split the dataset to 60% train set and 30% test set. We train XGBoost on the train set and test it on the test set. We report its performance on the three categories in Table IV. Results indicate that the best classification results were for the *Decision* label, the second best were for the *Supporting Facts*. The *Rationale* classification results were the worst.

TABLE IV: Multi-label classification evaluation

Model	Precision*	Recall*	F1 score*
Random Forest	**0.73**	0.51	0.60
XGBoost	0.67	**0.60**	**0.63**
KNN	0.60	0.50	0.54
* Micro-averaged			
XGBoost classification evaluation			
Label	**Precision**	**Recall**	**F1-score**
Decision	**0.76**	**0.69**	**0.72**
Rationale	0.62	0.41	0.49
Supporting Facts	0.64	0.68	0.66

C. Summary

These results indicate that the most challenging task is rationale classification, possibly due to the overlap of categories in the messages themselves, as seen in Figure 1. We find that the *Decision Tree* and *XGBoost* techniques are most promising, but further investigation is needed to provide satisfactory results.

IV. Structuring Rationale Information

This section will discuss our contribution on *structuring commit rationale information* in an ontological-based manner.

A. Decision/Rationale (DR) Graph

In our previous work [7] we defined a graph structure to explicitly represent commits, their labelling, and their relationships. For example, we related *conflicting* or *similar* commits together and represented the *rationale* for each commit as free text. Our current work is investigating the structure and classification of the rationale for one commit; we thus omit those inter-commit relationships here. Instead, from our insights from our labelling procedure (Section II-C), we now focus on rationale information at the *sentence level* where sentences can have multiple labels.

Table II shows the example commit in textual form with sentences colored according to their label, while Figure 2 represents that commit as a DR graph. A commit has the usual information like the commit hash and the date, and each commit is linked to its author and the individual sentences within the commit text. Each *Sentence* at the bottom of Figure 2 has a link to the *nextSentence* (to retain the structure of the commit), and are multi-typed with each label. Section IV-C explains how these classifications are inferred automatically through ontological semantics, which aids the analysis and reporting functionality of the Kantara framework.

B. Ontological Representation

The DR graph described in the previous section explicitly captures concepts and their relation. This makes it natural to

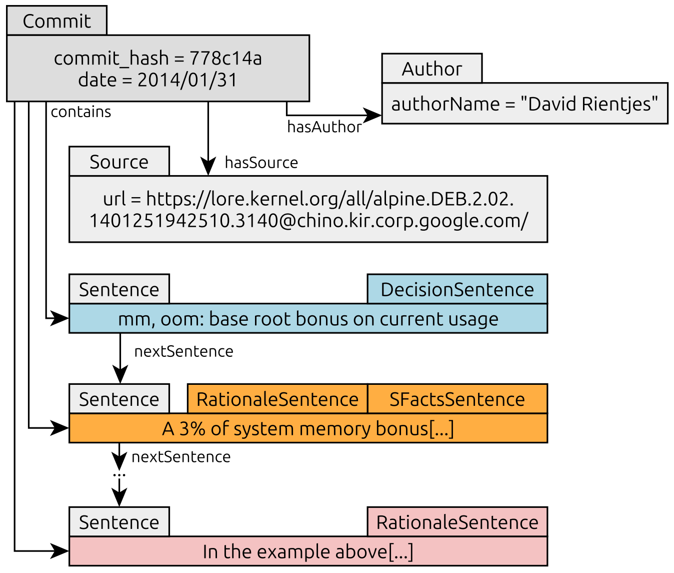

Commit
commit_hash = 778c14a
date = 2014/01/31
Author
authorName = "David Rientjes"
contains
hasAuthor
Source
hasSource
url = https://lore.kernel.org/all/alpine.DEB.2.02.1401251942510.3140@chino.kir.corp.google.com/
Sentence
DecisionSentence
mm, oom: base root bonus on current usage
nextSentence
Sentence
RationaleSentence
SFactsSentence
A 3% of system memory bonus[...]
nextSentence
Sentence
RationaleSentence
In the example above[...]

Fig. 2: Decision/Rationale Graph

opt for a representation of this DR graph using an *ontological basis*, as ontologies represent "a shared understanding of a domain" [13].

For the concrete ontological graph implementation, we use the openCAESAR framework[7] developed by the NASA Jet Propulsion Laboratory (JPL) instead of a traditional ontological tool such as Protegé. This is due to the approach of openCAESAR [9] which abstracts over the complex logical foundations of the well-known Web Ontology Language (OWL) to instead provide a high-level language for defining ontologies using the Ontological Modeling Language (OML).

Briefly, the openCAESAR methodology for creating ontologies focuses on the creation of a) *vocabulary models* which correspond to ontology concepts (the A-box), and b) *description models* which correspond to ontology individuals (the T-box). The Rosetta editor provided by the openCAESAR framework is shown in Figure 3. In Figure 3a, the textual OML syntax is shown for the *rationale* vocabulary, defining the core concepts for *Commits* and *Sentences* and their relationship. Listing 1) reproduces a portion of this vocabulary. Figure 3b presents the OML graphical syntax.

Listing 1: Excerpt of the OML rationale vocabulary model

```
aspect SentenceClassificationType
concept DecisionSentence <
SentenceClassificationType
concept RationaleSentence <
SentenceClassificationType
concept SupportingFactSentence <
SentenceClassificationType
relation entity SentenceClassification [
    from Sentence
    to SentenceClassificationType
    forward hasClassification
    asymmetric
]
```

[7]http://www.opencaesar.io/

On the right of Figure 3b are the types of sentence classification and the subtypes of sentences which connect to each of these types of classification. We leverage the ontological approach to assign these subtypes to individual sentences through *inferencing* discussed in Section IV-C.

The advantages are that a) subtypes of sentences can be automatically inferred by the ontological reasoner, and b) sentences can be labelled with multiple classification types. While this could be adequately captured using a standard meta-modelling approach, we found that this ontology-based approach simplified our analyses and more accurately captured the meaning of our labelling.

C. Inferencing

An advantage of storing the commit and rationale information in an ontology is that ontologies naturally form a graph structure (a "knowledge graph" or "knowledge base") suitable for inference and querying [8]. We leverage this inferencing capability of ontologies to perform additional reasoning on our data set. From the Rosetta editor, tasks are available to run the Pellet ontological reasoner to perform this inferencing and check the consistency of the vocabulary and descriptions.

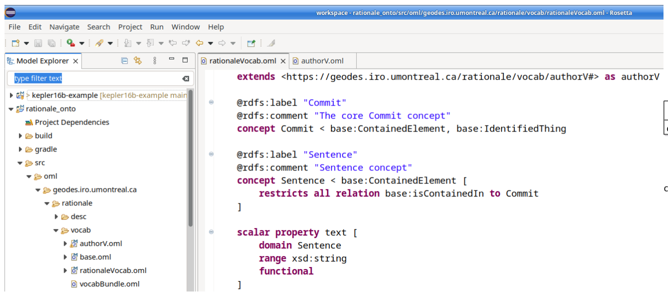
An example of this reasoning in the rationale vocabulary is found in Listing 2. The *rule* defines a antecedent/precedent *rule* where if *s* is a sentence and it has a classification of *ct*, where that *ct* is *Rationale*, then the sentence *s* should be inferred to be a *RationaleSentence*. While this is a straightforward rule, the benefit is that it simplifies the further analysis and reporting of which sentences contain rationale. That is, the analysis and reporting steps will not have to follow extra edges to determine if a sentence is classified as having *rationale* or not.

Listing 2: Classification rules for Sentences and Commits

```

concept DecisionSentence < Sentence
concept RationaleSentence < Sentence
concept SupportingFactSentence < Sentence

rule RationaleSentenceInfer [
    Sentence (s) & hasClassification (s, ct) &
    Rationale (ct) -> RationaleSentence (s)
]

concept CommitWithRationale = Commit [
    restricts base:contains to min 1
    RationaleSentence
]
```

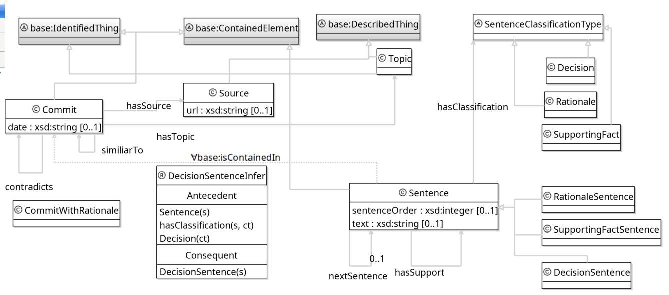
The *restricts* inference rule in Listing 2 goes farther with a similar typing action. Here, a commit that *contains* at least one *RationaleSentence* is a *CommitWithRationale*. Again, this simplifies analysis and reporting by removing the need to resolve links during those phases. For instance, without this inferencing, each analysis and report would have to determine if a commit contains a sentence with a classification type of rationale. Instead, this inferencing allows us to directly query for whether the commit is a *CommitWithRationale*.

Section V presents an analysis query and visualization which we have developed within the openCAESAR framework



```
extends <https://geodes.iro.umontreal.ca/rationale/vocab/authorV#> as authorV

@rdfs:label "Commit"
@rdfs:comment "The core Commit concept"
concept Commit < base:ContainedElement, base:IdentifiedThing

@rdfs:label "Sentence"
@rdfs:comment "Sentence concept"
concept Sentence < base:ContainedElement [
    restricts all relation base:isContainedIn to Commit
]

scalar property text [
    domain Sentence
    range xsd:string
    functional
]
```

(a) Textural format



(b) Graphical format

Fig. 3: Rationale vocabulary editing within the Rosetta component of openCAESAR

to better understand the rationale information. This query directly relies on these inferences.

V. Analysis

This section discusses the last step of the Kantara framework: analysis and visualization of the extracted rationale information. The intention of Kantara is to provide the user with insights into the presence of rationale in the commit sentences. This can be used to increase the quality of commit messages. For example, the patch submission process could require a certain percentage of the commit to contain rationale. Or, this could be used to identify authors or subsystems with insufficient rationale, so that interventions could be made.

A. Graph Queries

We employ the openCAESAR framework to build the rationale ontology, such that Kantara can produce OML *description* models through text generation which conform to this vocabulary (Section IV-B). Then, we can execute SPARQL queries on these description models to obtain results. For a developer to better understand the rationale information present in our DR graph, we have created queries such as querying for a list of authors and their commits, and querying for those sentences classified as *DecisionSentence*, *RationaleSentence*, or *SupportingFactSentence*.

Listing 3 demonstrates a more comprehensive SPARQL query (with omitted PREFIXes) which can be executed within the openCAESAR framework to return a JSON result containing commits, their authors, and whether each commit contains rationale. This table also contains the text and classification type(s) for each sentence in that commit.

Note that this query directly relies on the inference rules discussed in Section IV-C. With the typing provided by the inference rules, the query can be more concise. This motivates our choice of using an ontological approach for storing this rationale information.

B. Visualization

The results of the query in Listing 3 can be inspected in tabular form but we developed Kantara to also support interactive visualization, e.g., to display authors, their commits, and the sentences within those commits. The intention is for developers to use such visualizations to understand the rationale information present in each commit.

Listing 3: Query for commits and labelled sentences

```
SELECT ?author ?commit_id ?order ?text
(BOUND(?hasRationale) AS ?isCommitWithRationale)
(BOUND(?rct) AS ?isSentenceRationale)
(BOUND(?dct) AS ?isSentenceDecision)
(BOUND(?sct) AS ?isSentenceSupporting)
WHERE {
  ?commit a rationale:Commit .
  ?commit baseV:hasIdentifier ?commit_id .
  ?commit rationale:hasAuthor ?author_id .
  ?author_id authorV:authorName ?author .
  ?commit baseV:contains ?s .
  ?s rationale:text ?text .
  ?s rationale:sentenceOrder ?order
  OPTIONAL {
    ?commit ?hasRationale rationale:
    CommitWithRationale .
  }
  OPTIONAL {
    ?s ?rct rationale:RationaleSentence .
  }
  OPTIONAL {
    ?s ?dct rationale:DecisionSentence .
  }
  OPTIONAL {
    ?s ?sct rationale:SupportingFactSentence .
  }
}
ORDER BY ?commit_id ?order
```

Figure 4 shows a screenshot of a particular visualization that that centres committers' practices. In the example shown we focus on OOM developer *Michel Lespinasse*, who has authored one commit (*c25*) without rationale, and two commits (*c26*, *c27*) with rationale. The interactivity of the visualization allows the user to collapse or expand the children of nodes to focus on certain authors or commits. The visualization runs within a web browser and is built using Javascript and the D3 library, modified from an example provided in openCAESAR.

Authors are the left-most node in Figure 4 where the color indicates whether their commits contain rationale or not. Each commit (*c25*, *c26*, etc.) are shown to the immediate right. Finally, the sentences making up that commit are on the right-

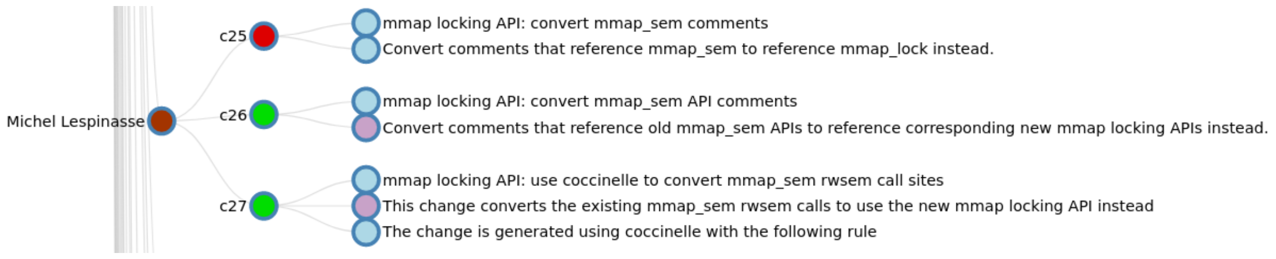

Michel Lespinasse
c25
c26
c27
mmap locking API: convert mmap_sem comments
Convert comments that reference mmap_sem to reference mmap_lock instead.
mmap locking API: convert mmap_sem API comments
Convert comments that reference old mmap_sem APIs to reference corresponding new mmap locking APIs instead.
mmap locking API: use coccinelle to convert mmap_sem rwsem call sites
This change converts the existing mmap_sem rwsem calls to use the new mmap locking API instead
The change is generated using coccinelle with the following rule

Fig. 4: Interactive visualization of authors, their commits, and commit sentence text

hand side, with the same coloring as in Figure 1 and Table II to indicate the classification of the sentence. In Figure 4, all seven sentences are labelled as *decisions*, and two are also labelled as *rationale*.

This visualization provides insight in committers' rationale documentation practices in these commits. It lets a developer understand the presence of decisions, rationale, and supporting facts written by others, exposing commonalities in a set of commits. It also highlights committers whose commits do not provide rationale. This could help the community to improve its rationale documentation practices, by alerting maintainers to commits with insufficient rationale information.

VI. Related Work

We divide related work into two categories: *extraction of rationale* and *representations of rationale*.

A. Extracting Rationale

Sharma *et al.* employed a heuristics-based approach to try to unearth rationale from Python Email Archives [14]. In [15], the authors propose ASGAR, a semantic grammar-based approach to automatically capture and structure design rationale. These works differ from ours as we propose an NLP-based approach.

Other researchers have employed machine learning (ML) techniques to extract rationale. In [16], the authors tried to automatically identify decisions from textual artifacts. They created a labelled ground truth from the Hibernate developer mailing list and tried various classification models and configurations. Similarly, in [17], the authors analyzed Apache commit messages to study the impact of message quality on software defect proneness. As a quality indicator, the authors considered the binary presence of rationale information and they manually labelled the commits as containing or not motivation/rationale information. These works employ binary classification, thus our work differs by proposing a rationale representation with different multi-classifications (Table I).

ML techniques have been employed in an attempt to solve the *capture* problem (i.e., the large effort required to capture the rationale manually and adhere to a specific representation) [18]. They aim at bridging the gap with the unused existing rationale representation models such as Burge's rationale model [1]. For example, in [19], the authors propose annotating chat messages that contain rationale by capturing five rationale elements adapted from the IBIS model [20]. In [5], the authors focus on designing an algorithm to extract and structure design rationale from design documents based on the ISAL model [21]. Rogers *et al.* investigated the usage of ontology and linguistic features to identifying rationale from Chrome bug reports [22], Lester *et al.* investigate evolutionary algorithms for optimal features to extract and classify design rationale from Chrome bug reports and design discussion transcripts [23], and Mathur tried to improve Lester's classification results for the rationale with new features and algorithms [24]. These model-based classification approaches differ from the data-driven categorization we employ in this work.

B. Representing Rationale

Recently, other rationale representations have been proposed. AlSafwan *et al.* proposed a model with 15 categories to represent rationale in code commits after conducting interviews and a survey [25]. Hesse *et al.* proposed a documentation model for decision knowledge built upon the results investigating the comments to 260 issue reports from the Firefox project [26]. Kleebaum *et al.* built upon Hesse's documentation model and proposed Condec tools to support requirements engineers in documenting and exploiting decision knowledge for change impact analysis [27]. ConDec tools build up and visualize a knowledge graph consisting of knowledge elements and trace links. Soliman *et al.* worked on an empirically-grounded ontology for architecture knowledge from StackOverflow [28]. Their work also supports automating architecture knowledge capturing and proposing solutions to effectively search for relevant architectural information in developer communities. Bhat *et al.* proposed AdeX, a framework to extract and reuse architecture knowledge and to recommend alternative architectural solutions that could be considered during architectural design making [29]. The authors propose a static and a dynamic architectural knowledge models. In AdeX, the rationale of a decision refers to the quality attribute the decision addresses. AdeX automatically identifies architectural elements in design decisions. These architectural elements are identified using concepts captured in a publicly available cross-domain ontology.

The related work differs from our in two main ways. First, none of these works focus on representing and managing the rationale specifically for the Linux kernel. Second, we provide a knowledge graph-based approach and pipeline using the state-of-the-art openCAESAR tool for constructing and querying ontologies.

VII. Conclusion and Future Work

This article has presented the Kantara framework for structuring and extracting the rationale found in the commits of developers. We have presented our new classification of rationale information, focusing at the sentence-level and now including *rationale*, *decision*, and *supporting fact* labels. We discussed approaches to classify each sentence using NLP approaches and a brief evaluation. Finally, we described how this rationale information is represented using an ontologically-based knowledge graph in the openCAESAR framework [9], along with the analysis and reporting capabilities in Kantara.

Challenges and Future Work

a) Challenge 1: Expanding the Rationale Ontology: Our categorization of components of the rationale of commits was data-driven where we came to a consensus after several discussions during labelling II-C. Our model here focuses on sentence-level labelling to evaluate our Kantara framework and the NLP classification. In the future, we plan to extend our representation to include all the components of the rationale of commit messages (e.g. *Goal*, *Need*, *Benefit*, etc. [25]) and other previous rationale representations (Section VI).

b) Challenge 2: Subjective labeling: It is possible that our manual labelling process has introduced unintentional bias. To address this, the three authors labelled independently. A Fleiss kappa of 0.69 indicates a high reliability of our labelling. In the future, we are exploring avenues such as: a) including kernel developers themselves to validate our labels, or b) determining if large language models (LLMs) can provide suitable performance for this labelling task.

c) Challenge 3: Classification performance: As shown in Section III, the performance of the classification approaches was rather average. While it may not be possible to achieve perfect performance due to the subjective nature of this classification, we are attempting to improve these metrics. Concrete steps include a) applying hyper-parameter tuning to all classifiers to find optimal hyper-parameters, b) expanding our labelled data set of commit messages, and c) investigating more advanced classification architectures such as Bidirectional Long Short-Term Memory (Bi-LSTM) [30] or pre-traineed language models, e.g., BERT [31]

d) Challenge 4: Generality: The last challenge we wish to address is the threat to validity of generalizing this classification approach and our insights. Section II describes how the Linux kernel and the OOM subsystem have a particular development culture which emphasizes detailed commit messages. This culture may not be present in other projects, such that developers are not encouraged or required to produce commits with such detailed rationale information. In future work, we will investigate this by applying our Kantara approach to other codebases.

Acknowledgment

This research work is partially funded by the Fonds de Recherche du Québec – Nature et Technologies (FRQNT) Doctoral Research Scholarship (B2X).

References

[1] J. E. Burge, J. M. Carroll, R. McCall, and I. Mistrik, *Rationale-based software engineering*. Springer, 2008.

[2] D. Noble and H. W. Rittel, "Issue-based information systems for design," 1988.

[3] J. Conklin and M. L. Begeman, "gIBIS: A hypertext tool for exploratory policy discussion," *ACM Transactions on Information Systems (TOIS)*, vol. 6, no. 4, pp. 303–331, 1988.

[4] J. E. Burge, "Software engineering using design rationale," Ph.D. dissertation, Worcester Polytechnic Institute, 2005.

[5] Y. Liang, Y. Liu, C. K. Kwong, and W. B. Lee, "Learning the "whys": Discovering design rationale using text mining—an algorithm perspective," *Computer-Aided Design*, vol. 44, no. 10, pp. 916–930, 2012.

[6] Y. Tian, Y. Zhang, K.-J. Stol, L. Jiang, and H. Liu, "What makes a good commit message?" in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 2389–2401.

[7] M. Dhaouadi, B. J. Oakes, and M. Famelis, "End-to-end rationale reconstruction," in *37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–5.

[8] B. J. Oakes, B. Meyers, D. Janssens, and H. Vangheluwe, "Structuring and accessing knowledge for historical and streaming digital twins," in *First Workshop on Ontology-Driven Conceptual Modeling of Digital Twins*, 2021, pp. 1–13.

[9] M. Elaasar, N. Rouquette, D. Wagner, B. J. Oakes, A. Hamou-Lhadj, and M. Hamdaqa, "openCAESAR: Balancing agility and rigor in model-based systems engineering," in *Proceedings of the System Analysis and Modelling (SAM) Conference, in the 26th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2023, accepted.

[10] M. Dhaouadi, "A data set of extracted rationale from Linux kernel commit messages," in *Proceedings of Foundations of Software Engineering (FSE)*, 2023, pp. 1–2. Accepted.

[11] L. Wang, *Support vector machines: theory and applications*. Springer Science & Business Media, 2005, vol. 177.

[12] T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, K. Chen, R. Mitchell, I. Cano, T. Zhou *et al.*, "XGBoost: extreme gradient boosting," *R package version 0.4-2*, vol. 1, no. 4, pp. 1–4, 2015.

[13] M. Uschold, "Knowledge level modelling: concepts and terminology," *The knowledge engineering review*, vol. 13, no. 1, pp. 5–29, 1998.

[14] P. N. Sharma, B. T. R. Savarimuthu, and N. Stanger, "Extracting rationale for open source software development decisions—a study of Python email archives," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1008–1019.

[15] R. McCall, "Using argumentative, semantic grammar for capture of design rationale," in *Design Computing and Cognition'18*. Springer, 2019, pp. 519–535.

[16] X. Li, P. Liang, and Z. Li, "Automatic identification of decisions from the hibernate developer mailing list," in *Proceedings of the Evaluation and Assessment in Software Engineering*, 2020, pp. 51–60.

[17] J. Li and I. Ahmed, "Commit message matters: Investigating impact and evolution of commit message quality," *2023 IEEE/ACM 45rd International Conference on Software Engineering (ICSE)*, 2023.

[18] J. E. Burge, "Design rationale: Researching under uncertainty," *AI EDAM*, vol. 22, no. 4, pp. 311–324, 2008.

[19] R. Alkadhi, J. O. Johanssen, E. Guzman, and B. Bruegge, "React: An approach for capturing rationale in chat messages," in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2017, pp. 175–180.

[20] H. Rittel and D. Noble, "Issue-based information systems for design," *Univ. Calif. Berkeley Work. Pap*, vol. 492, 1989.

[21] Y. Liu, Y. Liang, C. K. Kwong, and W. B. Lee, "A new design rationale representation model for rationale mining," *Journal of Computing and Information Science in Engineering*, vol. 10, no. 3, 2010.

[22] B. Rogers, Y. Qiao, J. Gung, T. Mathur, and J. E. Burge, "Using text mining techniques to extract rationale from existing documentation," in *Design Computing and Cognition'14*. Springer, 2015, pp. 457–474.

[23] M. Lester and J. E. Burge, "Identifying design rationale using ant colony optimization," in *Design Computing and Cognition'18*. Springer, 2019, pp. 537–554.

[24] T. Mathur, "Improving classification results using class imbalance solutions & evaluating the generalizability of rationale extraction techniques," Ph.D. dissertation, Miami University, 2015.

[25] K. Al Safwan, M. Elarnaoty, and F. Servant, “Developers’ need for the rationale of code commits: An in-breadth and in-depth study,” *Journal of Systems and Software*, vol. 189, p. 111320, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121222000668

[26] T.-M. Hesse, “Supporting software development by an integrated documentation model for decisions,” Ph.D. dissertation, Heidelberg University, 2020.

[27] A. Kleebaum, J. O. Johanssen, B. Paech, and B. Bruegge, “Continuous management of requirement decisions using the ConDec tools,” *Co-Located Events of the 26th International Conference on Requirements Engineering (REFSQ-JP’20)*, 2020.

[28] M. Soliman, M. Galster, and M. Riebisch, “Developing an ontology for architecture knowledge from developer communities,” in *2017 IEEE International Conference on Software Architecture (ICSA)*. IEEE, 2017, pp. 89–92.

[29] M. Mahabaleshwar, “Tool support for architectural decision making in large software intensive projects,” Ph.D. dissertation, Technische Universität München, 2020.

[30] P. Zhou, W. Shi, J. Tian, Z. Qi, B. Li, H. Hao, and B. Xu, “Attention-based bidirectional long short-term memory networks for relation classification,” in *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers)*, 2016, pp. 207–212.

[31] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.