Matching Service

Table of Contents

About arc42

arc42, the template for documentation of software and system architecture.

Template Version 8.2 EN. (based upon AsciiDoc version), January 2023

1. Introduction and Goals

This document describes the Matching Service, which is used by correlators to match rules with messages. In order to also match faulty messages (e.g. Voice-Messages which are transcribed) with the best rule, this service offers multiple matching methods like phonetic, lexicographic or ML algorithms using openai. For each correlator a new instance of this service should be easily initialized, configured and even upgraded to match the different use cases of the correlator.

1.1. Requirements Overview

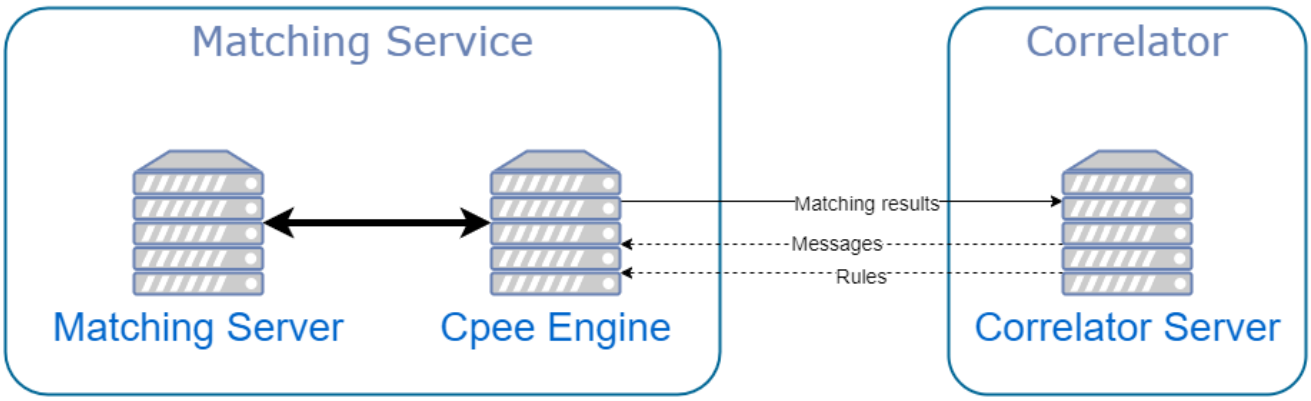
Functional Requirement	**Explanation**
Matching	
Data retrieval	Creating connections with the correlator in order to retrieve new messages and rules.
Data deletion	Deletion of already matched messages or rules.
Data storing	
Sending back matching results	
Combining matching-results	Combining the matching results of multiple methods and deleting duplicate matches to the same rule.

1.2. Quality Goal

Quality Goal	**Explanation**
Configurability	Ease of configurability of the service by the user for different correlators and use cases.
Upgradeability	Ease of upgradeability of the service to add more functionality like new matching methods or more complex deletion schemes.
Performance	The service should return results in a reasonable time (max. few seconds) as soon as new data comes in.
Transparency	Ease of understanding, seeing and testing the service and it’s underlying data flows.

2. System Scope and Context

2.1. Technical Context



The data objects must have the following properties (this is the bare minimum, for additional functionality more properties can be added):

Object	Properties
Rule	**listen_to** (String): Phrase or keywords on which the rule will be matched with the messages. **callback** (String): Callback-url the matched message will be send to. **id** (String): Unique identifier of the rule.
Message	**message** (String): Body of the message. **id** (String): Unique identifier of the message.
Matching result	**message** (String): Body of the message that was matched with the rule. **callback** (String): Callback-url the matched message will be send to.

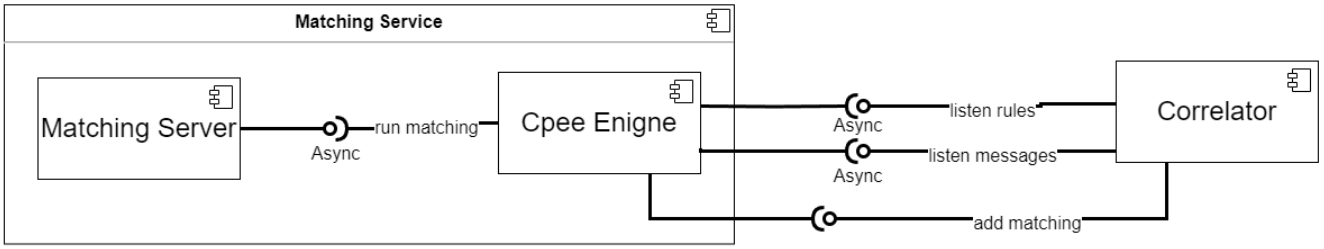
3. Solution Strategy

The service has a service-oriented architecture, implemented with the workflow execution engine Cpee as the orchestrator of subservices. All matching related tasks are implemented on a Python server, other tasks, mainly related to data management, are implemented with Ruby scripts directly inside Cpee. By using a workflow execution engine the service is uncomplicated to configure as we can offer a UI with an interactive BPMN diagram in which the user can build his own setup. Additionally, we can simply add functionality in form of subservices which can be called by the Cpee instance. We also achieve a high transparency as users can watch through the UI of Cpee which tasks are executed and what data is stored.

4. Building Block View

4.1. Whitebox Overall System

The following figure shows the internal top-level decomposition of the Matching Service.



Cpee Engine

is the orchestrator of the matching service. It receives and saves the rules and messages from the Correlator, calls the different matching methods, manages the matching results and sends them back to the Correlator.

Matching Server

is a Python server responsible for implementing and initiating matching instances with different matching methods.

Correlator

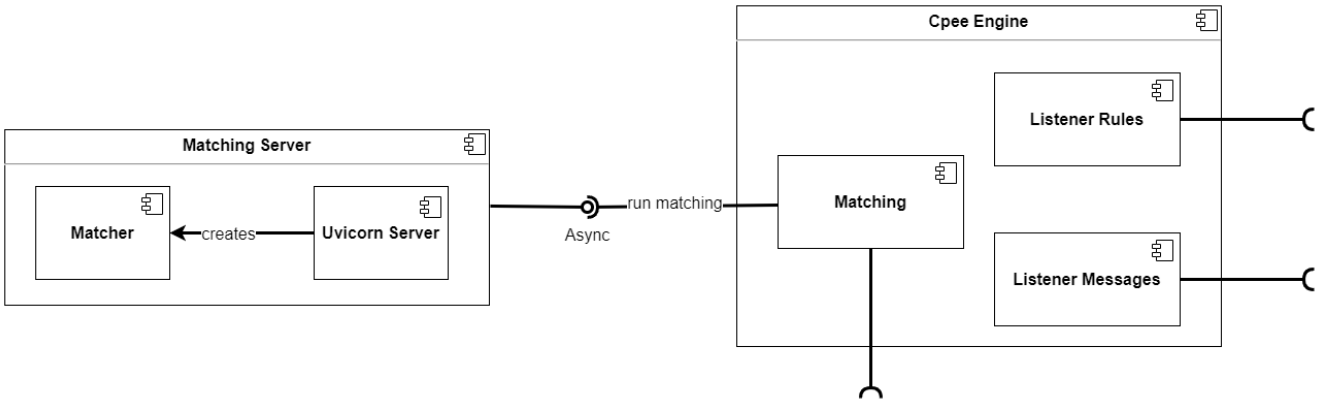
receives rules and listens to messages from a messaging service. Whenever new data comes in, it sends them to the Matcher Service and returns the matching results to their individual callbacks.

Important Interfaces

run matching
enables the Cpee engine to specify the matching method and to send all the needed data to the Matching Server.

4.2. Level 2

The following figure shows the internal structure of the Matching Server and the Cpee Engine.



Listener Rules

manages asyncronous connection with Correlator and saves new rules to the temporary storage of the Cpee Engine.

Listener Messages

manages asyncronous connection with Correlator and saves new messages to the temporary storage of the Cpee Engine.

Matching

is the core piece of the Cpee Engine. When changes in the data are recognized, the Matching Server is called. The resulting matching results are combined and sent to the Correlator. Lastly, rules and messages in the matching results are deleted from the temporary storage.

Uvicorn Server

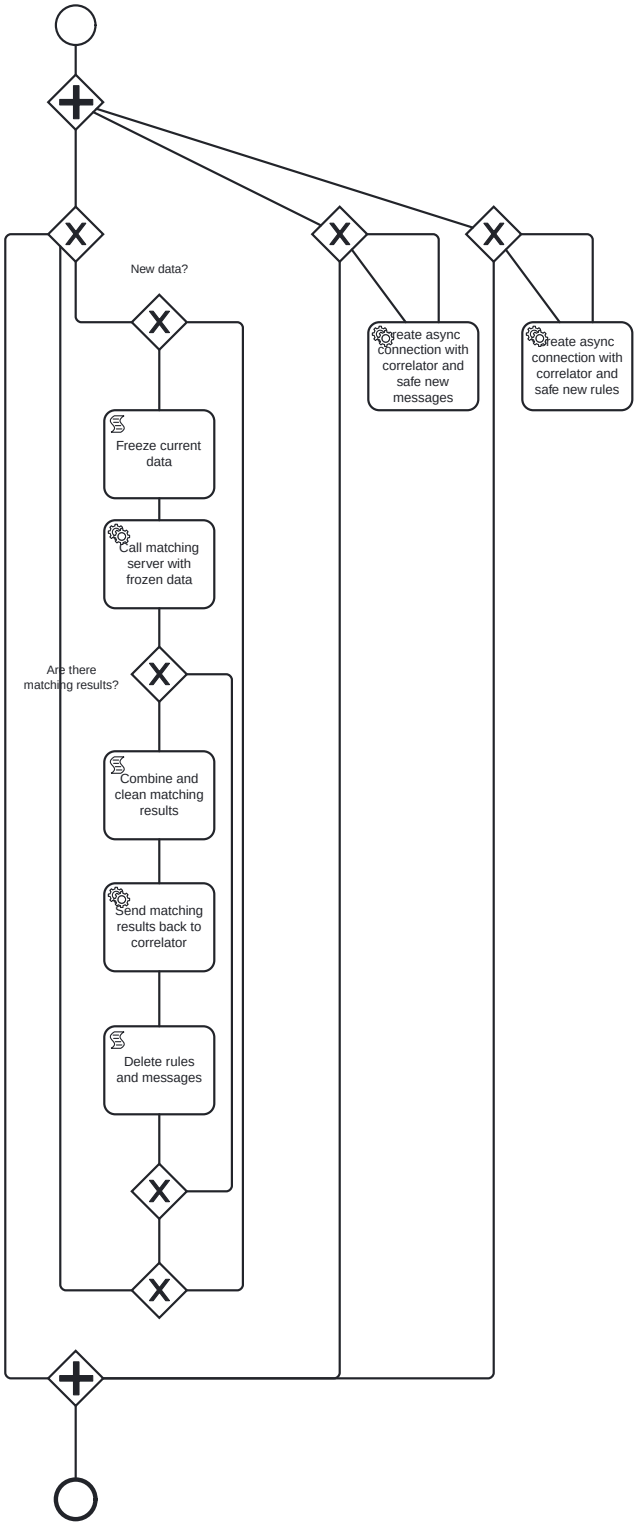
offers the interface for the Cpee Engine to send the needed data to start a Matcher.

Matcher

matches the rules and the messages with the given method and returns the results to the Cpee Engine.

5. Runtime View

The following figure is a redrawing of the BPMN-diagram used in the Cpee Engine:



5.1. Example: A new rule gets inserted

- The service call that established the asynchronous connection with the Correlator receives a new rule and pushes it to the back of the rule list in the local storage of the Cpee engine.
- The matcher thread recognizes that the length of the rule list changed and starts the matching process:
  1. It freezes the current rule and message list in a new variable.
  2. It calls the matching server with the frozen data to start a matching with the phonetical method.
  3. It calls the matching server with the frozen data to start a matching with the lexicographical method.
  4. If there are matching results, they get cleaned up by deleting duplicate matches with the same rule.
  5. For each match the callback-url of the rule and the message-body of the message are sent as tuples to the correlator.
  6. Rules and messages found in the matching result are deleted from the rule and message list.
  7. The new length of the rule and message list are calculated to recognize new changes to the lists.

6. Risks and Technical Debts

- The performance may suffer with large amounts of rules and messages, as the matching will always be run on the whole dataset as soon as only one entry is added. Consequently, the latency (the time between the addition of data and getting back the matching results) for the correlator may be very high.
- Data is only temporarily stored on the Cpee engine as soon as there is a failure on the Cpee engine it could be that the data is lost with no possibility to recover it.

7. Possible Ugrades

- Performance optimizations (e.g. by only matching new rules and new messages), as currently all messages and rules are matched as soon as new data comes in
- More intricate deletion scheme for rules and messages (e.g. deletion after time limit, a limit of matchings or never)
- Better filtering of matching results (e.g only taking the best matches), as currently the first match is always the winner
- More matching methods
- A more configurable unifying process of matching results from different methods (e.g. possibility of prioritizing specific matching methods)

8. Links

- Github project: https://github.com/SebaAeEr/Matching-Service
- Cpee: https://cpee.org/