FUSION: Feature-based Processing of Heterogeneous Documents for Automated Information Extraction

Michael Sildatke[1], Hendrik Karwanni[1], Bodo Kraft[1] and Albert Zündorf[2]
[1]*FH Aachen, University of Applied Sciences, Germany*
[2]*University of Kassel, Germany*

Keywords: Architectural Design, Refactoring and Patterns, Model-driven Software Engineering, Process Modeling, Quality Management, Software and Systems Modeling, Enterprise Information Systems, Information Extraction, Document Classification, Feature Detection, Software Metrics and Measurement.

Abstract: Information Extraction (IE) processes are often business-critical, but very hard to automate due to a heterogeneous data basis. Specific document characteristics, also called features, influence the optimal way of processing. **Ar**chi**t**ecture for Automated Generat**i**on of Distributed In**f**orm**a**tion Extra**ct**ion Pipelines (ARTIFACT) supports businesses in successively automating their IE processes by finding optimal IE pipelines. However, ARTIFACT treats each document the same way, and does not enable document-specific processing. Single solution strategies can perform extraordinarily well for documents with particular traits. While manual approvals are superfluous for these documents, ARTIFACT does not provide the opportunity for Fully Automatic Processing (FAP). Therefore, we introduce an enhanced pattern that integrates an extensible and domain-independent concept of feature detection based on microservices. Due to this, we create two fundamental benefits. First, the document-specific processing increases the quality of automated generated IE pipelines. Second, the system enables FAP to eliminate superfluous approval efforts.

1 INTRODUCTION

In many businesses, heterogeneous and human-readable documents are the input of underlying Information Extraction (IE) processes. Due to an unstructured data basis, classic Extract, Transform, Load (ETL) technologies reach their limits. Environments may change rapidly so that the number of different formats increases over time. Flexible architectures are the basis for businesses to react to those changes at an early stage.

The IE processes include different tasks, like document conversion, element decomposition and information extraction. Single software components can solve specific tasks, while their composition results in document processing pipelines. **Ar**chi**t**ecture for Automated Generat**i**on of Distributed In**f**orm**a**tion Extra**ct**ion Pipelines (ARTIFACT) provides an architecture that enables the separation of concerns and the automated generation of IE pipelines. Due to automatic quality determination mechanisms, the architecture finds the global best pipelines for extracting specific information on its own.

But in practice, specific document characteristics, also called features, can limit the set of possible solution strategies. If a document does not contain tables, e.g., any table-based extraction strategies will not be suitable for that document. In this case, the global best pipeline does not need to be the optimal document-specific pipeline. Therefore, opportunity costs may arise if the system does not consider these document features during automated pipeline generation. Furthermore, pipelines can reach excellent results for documents with specific characteristics. In these cases, manual checks at the end of the automatic process are superfluous, and Fully Automatic Processing (FAP) is possible.

This paper extends the ARTIFACT pattern to enable the system to consider critical features during the automated pipeline generation. This way, we ensure that the provided system always generates the best document-specific pipeline. Furthermore, we empower the provided system to FAP of documents. Hence, our extension increases business value and eliminates superfluous efforts for manual approvals.

The paper is structured as follows: Section 2 describes the project that motivates our approach. Section 3 describes related works, while Section 4

Sildatke, M., Karwanni, H., Kraft, B. and Zündorf, A.
FUSION: Feature-based Processing of Heterogeneous Documents for Automated Information Extraction.
DOI: 10.5220/0011351100003266
In *Proceedings of the 17th International Conference on Software Technologies (ICSOFT 2022)*, pages 250-260
ISBN: 978-989-758-588-3; ISSN: 2184-2833

presents the key concepts of ARTIFACT as the architectural basis. Section 5 introduces **F**eat**u**re Ba**s**ed Process**i**ng of Heterogeneous Documents for Automated Information Extracti**on** (FUSION). Section 6 describes the experimental evaluation of FUSION in the real-world project, while Section 7 summarizes the paper and gives an overview of future work.

2 MOTIVATION

The introduced pattern is domain-independent and especially applicable to any domain with a lack of document standards. The German energy industry serves as an example to motivate the extension of ARTIFACT. Due to its microservice-based architecture and generic concepts, the approach ensures extension of scope and the adaptability for other domains.

Service providers collect product information from about 3,150 energy suppliers with 15,000 different electricity or gas products. Since there is no standard, the published documents containing information about these products have various formats. The document basis, therefore, is heterogeneous, and service providers mostly have to extract this information by hand, e.g., product prices.

ARTIFACT handles every document equally and does not consider any document-specific characteristics. But in practice, these characteristics, also called features, are metadata that can influence the optimal way of processing. While these features are mostly irrelevant for human extractors, they are critical for automated IE. Depending on these features, the processing system should choose different solution strategies. Thus, knowledge about these features and their consideration during pipeline generation can strongly support the success of automated IE.

Documents can have technical (domain-independent), and content-based (domain-specific) features. The following list contains typical technical features of documents:

- **Presence or absence of price tables.** If price tables are present, pipelines will have to integrate specific table decomposition components. Also, they need particular extractors that have table elements as input. In case of the absence of tables, the pipeline should integrate any non-table-based extractors.
- **Scanned or screenshot documents.** If a document is scanned or screenshot, specific preprocessing steps will be necessary. These steps can, e.g., be content aligning or OCR-processing.
- **Page segmentation.** If the document has different columns organizing the content, e.g., if the document is a booklet, pipelines will have to integrate specific segmentation decomposers to keep the correct content ordering.

There are also different content-based features. In the case of the underlying research project originating from the German energy industry, the following list contains an exemplary set of typical ones:

- **The number of listed products.** A document can list one or more products. If there is only one product, it is clear where the presented information belongs. If there are several products, information has to be related to the correct product.
- **Price representations.** Prices can have different representations. While gross and net prices are common in every domain, there can be more domain-specific representations, e.g., net excluding transportation. Automated extractors have to understand that all representations belong to the same price.
- **Consumption-based prices.** Products can have consumption-based prices. If customers consume 2.500 kWh per year, e.g., they will have to pay 25.45 ct/kWh. If they consume more than 2.500 kWh per year, they will have to pay 26.34 ct/kWh. Hence, extractors have to extract additional information besides the actual prices.
- **Time-variable prices.** Products can have time-variable prices. The price for a consumed kWh can be 26.34 cents between 06:00 and 22:00, e.g., and 25.54 cents between 22:00 and 6:00. Automated Extractors have to recognize these time windows to extract prices correctly.
- **Presence or absence of previous prices.** Some suppliers provide previous prices belonging to a product to have a price comparison. These prices are irrelevant, and automated extractors have to ignore them during extraction.
- **Presence or absence of regional limits.** Some suppliers limit their products to specific regions. The presence of this information may require additional extraction steps.

Based on related features, there are certain document groups. All members of a particular group have similar characteristics (cf. Figure 1).

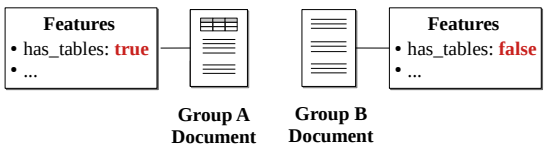

Figure 1: Feature-Based Document Group Examples.

Related document features can influence the optimal way of processing. A *group A document*, e.g., requires table-based, while a *group B document* requires text-based processing strategies.

Furthermore, specific document groups may have the potential of FAP because particular strategies may perform extraordinarily well on them. If a document only lists one product in a simple price representation, e.g., less complex processing strategies may always achieve correct results.

Hence, the automatic detection of document features can enable document-specific processing and supports the automated IE from very heterogeneous data. Additionally, the knowledge about certain features can enable FAP.

3 RELATED WORK

The challenges mentioned are primarily related to the fields of Business Process Automation (BPA), document classification, IE, as well as agile development.

Business Process Management (BPM) is a discipline that combines knowledge from information technology and management science to organize business processes efficiently (van der Aalst, 2004). Workflow Management (WFM) systems (Jablonski and Bussler, 1996) or Process-Aware Information Systems (PAISs) (Dumas et al., 2005), e.g., focus on the best operational support for BPM.

Modern techniques like Robotic Process Automation (RPA) support the automation of business processes. RPA replaces human interactions with robots by automating manual inputs into the Graphical User Interface (GUI) of an enterprise system (Ivančić et al., 2019). In RPA systems, robots learn rule-based behaviors by observing users interacting with affected systems (Aguirre and Rodriguez, 2017).

However, the underlying problem deals with an infinite set of possible document formats, so rule-based solution strategies are insufficient.

The developed system needs the ability of document classification to aim at document-specific processes. Document classification has a long history, and the first findings originate from the 1960s (Borko and Bernick, 1963, e.g.). In the area of text classification, it deals with the automated assigning of categories to a text document based on specific terms, also called features (Sebastiani, 2002).

Different algorithms solve feature-based text classification, e.g., k-nearest-neighbor (Jiang et al., 2012) or support vector machines (Lilleberg et al., 2015). However, these approaches and algorithms deal with problems in which the detection of features is less relevant and the classification itself is the focus.

Furthermore, the presented system must detect features from non-machine-readable PDF documents to solve the underlying problem. Also, the detection of (non-)text-based features is relevant for determining critical characteristics of a document, e.g., the presence or absence of tables. Therefore, the introduced approach deals with problems in which the feature detection itself is the most relevant task.

IE deals with extracting structured information from unstructured text (Cardie, 1997) and ETL processes focus on the automation of extraction pipelines (Albrecht and Naumann, 2008). Classic ETL technologies especially can handle structured data (Skoutas and Simitsis, 2006). (Schmidts et al., 2019), e.g., introduce an approach to process semi-structured tables in a well-defined format. Since the underlying problems deal with the processing of non-machine-readable PDF documents, such approaches are not sufficient. Furthermore, Ontology-Based Information Extraction (OBIE) systems focus on the IE from unstructured text (Saggion et al., 2007), but are not designed for the IE from PDF documents (Wimalasuriya and Dou, 2010).

The combination of document classification and IE can support implementing more efficient automation processes (Liu and Ng, 1996). Characteristics of features may change over time or new features can get relevant for classification tasks. Furthermore, ongoing research leads to many upcoming approaches for classification or IE tasks. In table analysis, e.g., several hundred approaches arose in 2019 (Hashmi et al., 2021). Agile development approaches like Extreme Programming or Scrum allow fast prototyping of new approaches and form the basis for flexible development processes (Beck, 2003; Rubin, 2012). Additionally, these approaches can ensure short feedback loops and form the basis for continuously measuring business-relevant quality metrics (Fowler, 2013).

Emergent architectures can enable evolution by adding code to ensure the adaption of new situations (Hanson and Sussman, 2021). Also, concepts like Domain-driven Design (DDD) require modular software (Tarr et al., 1999; Evans and Evans, 2004). (Newman, 2015) points out that microservices and Microservice Architecture (MSA) can form a basis for implementing modular and flexible software.

Derived from the aforementioned findings, the introduced approach uses ARTIFACT as its basis. ARTIFACT provides a microservice-based architecture that integrates business metrics and the flexible design of IE pipelines (Sildatke et al., 2022). Section 4 describes its concepts and the potential for extension resulting from the underlying problem.

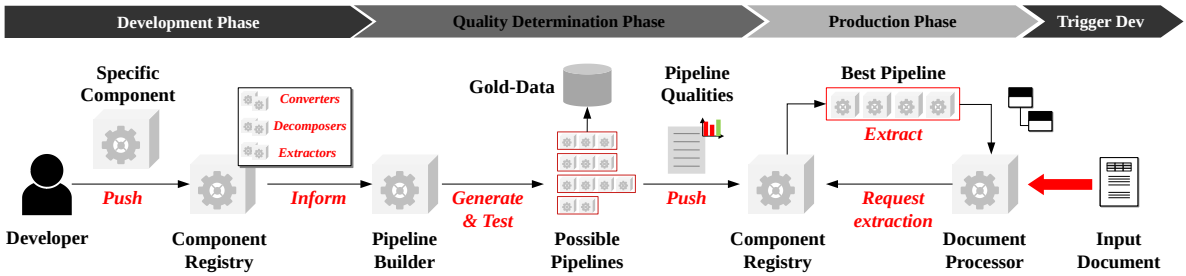

Figure 2: Overview of the ARTIFACT Pattern.

4 ARTIFACT PATTERN

Figure 2 shows an overview of the core building blocks of the ARTIFACT pattern.

In ARTIFACT, a pipeline consists of converters, decomposers and extractors, each representing a microservice. An extractor is always the last component of a pipeline and its result is specific information. Developers can develop new components and push them into the component registry, which manages all available components.

The pipeline builder can build all possible pipelines automatically with available components. It also tests all potential pipelines against a set of gold documents to determine the quality of each potential pipeline. The component registry can extract specific information in the production phase with the best available pipeline based on the determined qualities. Every time the component registry has an update, it will initiate the quality determination, e.g., triggered by the push of a new component.

Since the test data has to represent real-world conditions, there may be constellations in which a specific document type dominates, e.g., documents containing tables. In this case, the determination of the global-best pipeline tends to select a suitable one for the dominating document type (cf. Figure 3).

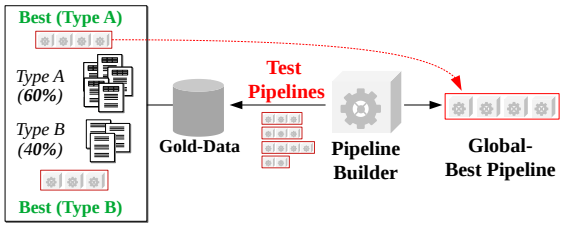

Figure 3: Biased Determination of Global Best Pipelines.

Due to this, executing the global best pipeline may not always be the optimal document-specific solution. Suitable document-specific pipelines may exist, while the systems will not take them into account following the concept of ARTIFACT (cf. Figure 4)

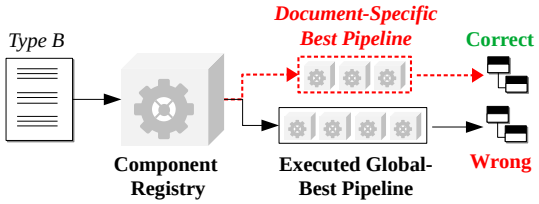

Figure 4: Document-Specific Best Pipeline Not Picked.

In ARTIFACT, a higher-level BPMN process controls the overall IE process. The last task of this process is a manual approval step that makes human interaction mandatory. Due to this process configuration, the implementation of FAP without any manual effort is not possible. To address these challenges, the provided approach extends the ARTIFACT pattern. It empowers the system to generate document-specific pipelines and creates the ability for FAP.

5 FUSION PATTERN

The following section introduces the FUSION pattern as an extension of ARTIFACT, while Figure 5 shows the building blocks. FUSION enables automated feature detection for input documents by providing the feature detector registry microservice. The label extender and feature matcher microservices ensure the automated generation of best document-specific pipelines. Based on the ARTIFACT core, developers can continuously extend the system.

Hence, it combines the concepts of document-specific processing and feature-based quality determination. The following subsections introduce the core concepts of FUSION.

5.1 Label Extension

Since document features are relevant for generating the optimal document-based pipeline, they have to be part of gold documents. To take features into account

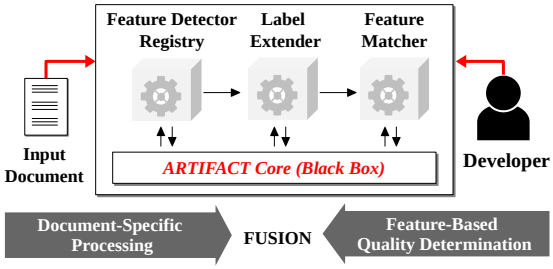

Figure 5: Core Components of FUSION.

when determining pipeline qualities, we manually extend the base gold standard data as shown in Figure 6.

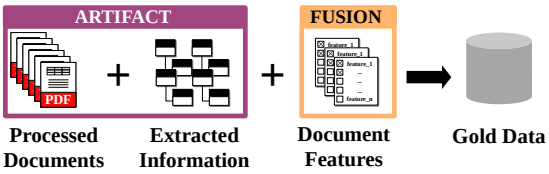

Figure 6: Feature Labels in Gold-Standard.

To find appropriate document groups, the system requires information about suitable pipelines for each gold standard document. To achieve this, FUSION adapts the pipeline quality determination mechanism of ARTIFACT to automatically extend the set of gold data with required pipeline labels (cf. Figure 7).

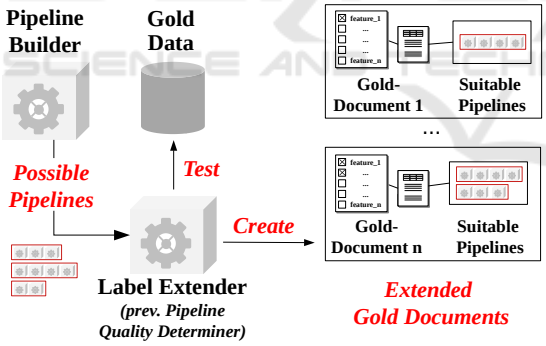

Figure 7: Result of the Label Extender.

To extend the set of gold data, the pipeline builder generates all possible pipelines and sends them to the label extender. Afterwards, the label extender tests each pipeline against each gold document. As a result, it automatically creates a list of extended gold documents, while each extended gold document now stores a list of suitable pipelines.

5.2 Feature Matching

The overall goal of the feature matcher is to find the best feature-based pipelines. In other words, it determines which pipeline the system will have to choose in production if an unknown input document has a specific set of features. Therefore, the feature matcher matches relevant features from extended gold documents to an ordered set of optimal pipelines (cf. Figure 8).

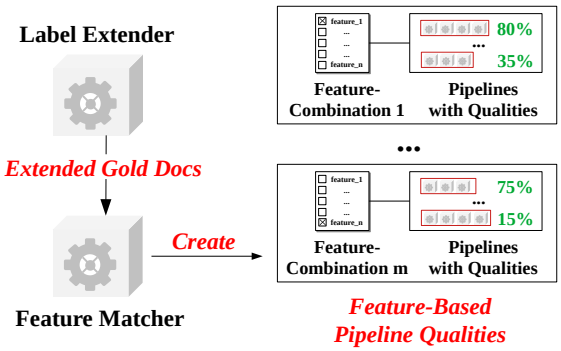

Figure 8: Result of the Feature Matcher.

Due to adaptability, developers can implement any suitable matching strategy, depending on the specific problem. In the introduced project, the feature matcher implements a rule-based strategy that builds all possible and unique combinations of boolean features.

If it knows the relevant features, the system will be able to find the optimal pipeline for an unknown input document using feature matching.

5.3 Feature Detection

FUSION introduces automated feature detection to determine the relevant features of an input document.

To achieve this, developers can implement different feature detectors that detect the value of a specific feature for any input document (cf. Figure 9).

Figure 9: In- and Output of a Feature Detector.

A feature detector is a microservice that consumes a specific type of document and detects the value of a particular feature, e.g., the presence or absence of tables in a PDF document.

FUSION also introduces a feature detector registry, where developers can deploy feature detectors. The feature detector registry manages all available feature detectors and ensures that a registering detector meets all quality requirements. For this, the registry requests the feature detector quality determiner (cf. Figure 10).

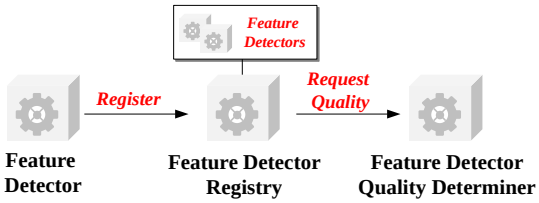

Figure 10: Registration to the Feature Detector Registry.

Domain experts or managers can define the necessary quality criteria, as shown in Table 1.

Table 1: Quality Criteria for Relevant Features.

Information	Required Quality
GLOBAL	*85%*
Specific Criteria for Feature 1	90%
...	...
Specific Criteria for Feature n	75%

A GLOBAL setting defines the required quality for all features that experts can overwrite to define specific criteria for particular features if necessary.

To determine the quality of a specific feature detector, the feature detector quality determiner tests this detector against each gold document (cf. Figure 11).

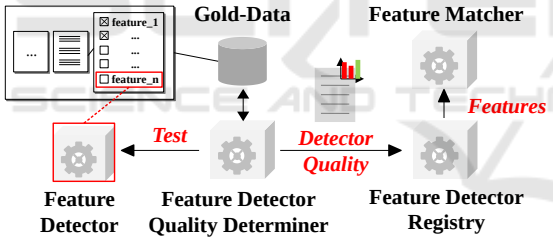

Figure 11: Quality Determination of Feature Detectors.

Afterwards, the determiner responds with the specific detector quality so that the registry can check if the detector meets the required criteria. If it reaches the requirements, the detector will be registered to the registry to be available for specific feature detection. Finally, the feature detector registry informs the feature matcher about the available features. The feature matcher only considers features that a corresponding detector can detect reliably.

This approach enables the rapid creation of new feature detectors or the improvement of existing ones.

5.4 Feature-based Document Processing

To be able to use document-specific pipelines, FUSION introduces the concept of feature-based document processing. As a basis for that, the feature matcher sends the information about feature-based pipeline qualities to the component registry (presented with ARTIFACT) after each matching, triggered by the registration of new feature detectors.

The document processor requests the feature detector registry to generate a feature-enriched document to achieve feature-based processing (cf. Figure 12).

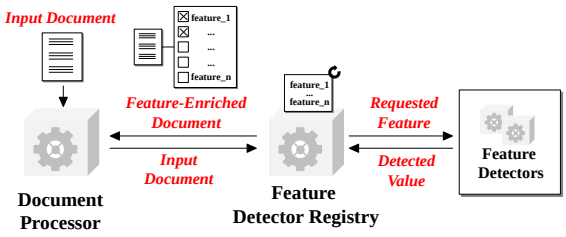

Figure 12: Generation of a Feature-Enriched Document.

The component registry can now find the best feature-based pipeline for a processed document (cf. Figure 13).

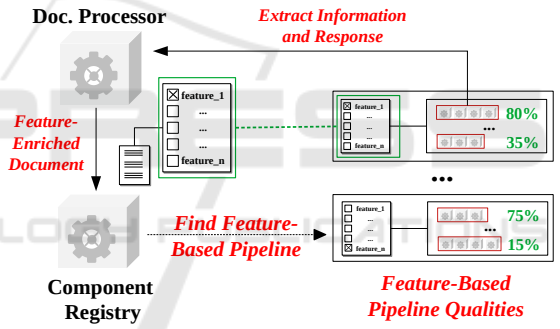

Figure 13: Feature-Based Document Processing.

The component registry maps the document features of an incoming document with the list of feature-based pipeline qualities and executes the best performing pipeline.

5.5 Result Routing

To realize FAP, FUSION introduces result routing. The basis for result routing is a configuration with defined criteria for FAP (cf. Table 2).

Table 2: Document Processor Configuration.

Information	FAP-Limit
GLOBAL	*95%*
Specific Criteria for Information 1	90%
...	...
Specific Criteria for Information 2	100%

Experts can define the *GLOBAL* criteria to treat all information the same way. In the case of a need to adjust the limit for specific information, e.g., if it is more or less business-critical, they can overwrite the GLOBAL configuration. If the extraction of all information meets the required FAP criteria, the system will route the document to FAP. Figure 14 shows an overview of the FUSION architecture, including the concept of result routing.

6 EXPERIMENTAL EVALUATION

In this section, we demonstrate the practical application of our FUSION pattern in the project motivated in Section 2. However, the approach is extensible and highly adjustable. Therefore, it allows flexible scope extension and the adaption to other domains. Since the introduced approach focuses on the automated detection of document features, we do not describe developed IE components further.

6.1 Features & Configuration

The data basis is very heterogeneous, and we identified an initial set of several key document features that characterize an input document. According to the defined features, we developed several feature detectors. Using the FUSION architecture, the system automatically tests each feature detector against the set of gold documents. Table 3 shows the quality of the developed detectors.

Table 3: Achieved Feature Detector Qualities.

Feature	Detector Quality
HAS_TABLES	86%
IS_SCREENSHOT	56%
IS_COLUMN_SEPARATED	45%
HAS_EXACTLY_ONE_PRODUCT	86%
HAS_GROSS_PRICES	91%
HAS_NET_PRICES	91%
HAS_OTHER_PRICE_REPRESENTATIONS	87%
HAS_STAGGERED_PRODUCTS	92%
HAS_TIME_VARIABLE_PRODUCTS	92%
HAS_REGIONAL_CONDITIONS	45%

We reached the required quality for seven features during the evaluation. Therefore, the system considers seven relevant features when generating feature-based pipelines.

We defined a global FAP-limit of 95% for all information. The document processor, therefore, will route each input document for FAP reaching this limit.

6.2 Achieved Results

Following the simple rule-based generation of all possible feature combinations with seven features leads to $2^7 = 128$ feature combinations. Due to specific domain knowledge about mutual exclusive characteristics, we know that the number of feature combinations is much smaller in reality. Hence, the rule-based feature matcher determines a set of 35 unique feature combinations in 1300 gold documents, while each feature combination has at least 30 related documents.

Table 4 shows the comparison of IE results using the global-best pipeline (ARTIFACT) and the feature-based pipeline (FUSION). The comparison indicates that feature-based pipelines increase quality without adding new extraction-specific components.

Table 4: Comparison of IE Results.

Information	∅ARTIFACT	∅FUSION	± %-Pts.
DateOfValidity	91%	91%	+ 0
BasicPrices	59%	77%	+ 18
CommodityPrices	54%	74%	+ 20
CustomerGroups	55%	84%	+ 29
MeteringPrices	34%	66%	+ 32
ProductType	55%	82%	+ 27
ProductCategory	55%	86%	+ 31

The project is still at an early stage, so improvements in converters, decomposers and extractors will also improve the results in the future.

However, the system detected six feature combinations for which specific pipelines perform extraordinarily well for all requested information. Since the extractions fulfill all FAP criteria in the test, the system will route input documents for FAP, if they have one of the six feature combinations. 174 documents belong to one of the six feature combinations, so we identified a potential for FAP of about 13% in gold data.

During the productive use of the FUSION pattern, the system processed 524 unknown documents and automatically related 76 of these 524 to one of the six feature combinations fulfilling the criteria for FAP. The documents were routed to FAP, while an audit showed that all results were correct. Hence, we aim at a quote of 14% FAP in practice.

6.3 Implementation

In the following subsection, we present an exemplary implementation of our FUSION architecture pattern.

Due to the extensibility and flexibility of the ARTIFACT reference implementation, we decided to build on this implementation. Therefore, we focus on

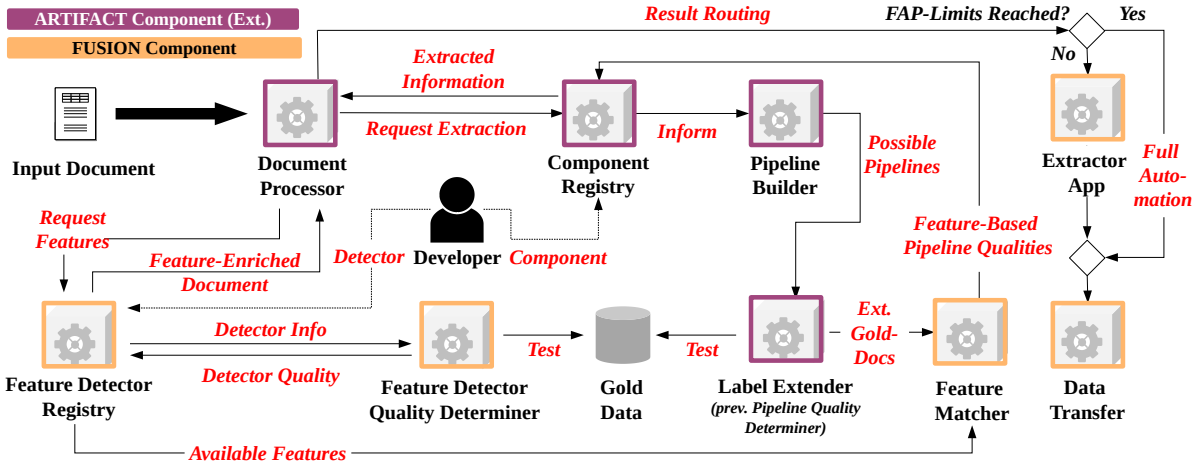

Figure 14: Overview of the FUSION Architecture.

the new and modified services and only briefly mention unchanged services from ARTIFACT.

Our current implementation of FUSION utilizes microservices encapsulated in Docker[1] images. We again realize the communication between microservices via Representational State Transfer (REST) calls and heavily rely on OpenAPI[2] to generate server stubs and client SDKs.

6.3.1 Feature Detectors

As stated in Subsection 5.3, feature detectors are used to detect a specific feature inside a document. Analogous to components in ARTIFACT, we implemented every feature detector as a microservice. Due to this, developers can choose the most suitable language and framework for a specific detection problem.

Each feature detector microservice offers two endpoints. The first endpoint provides information about the detector, e.g., its name and what type of feature it can detect. The second endpoint performs the detection, receiving a document and returning a boolean flag that indicates whether the document contains the feature.

6.3.2 Feature Detector Quality Determiner

As also mentioned in Subsection 5.3, we need to assess the quality of each feature detector and, therefore, implemented a FastAPI[3] webservice that tests a detector against the set of gold documents by detecting the respective feature in each gold document. The determiner calculates the resulting quality by dividing the number of correct results by the number of gold documents.

[1]https://www.docker.com/
[2]https://swagger.io/specification/
[3]https://fastapi.tiangolo.com/

6.3.3 Feature Detector Registry

As stated in Subsection 5.3, we implement a microservice that manages the feature detectors. This microservice utilizes a Java 17 stack with Maven and Spring Boot[4]. After we provide the required feature quality criteria mentioned in Subsection 6.1, we can start registering detectors at the feature detector registry. The registration process is mostly analogous to that of the component registry of ARTIFACT.

Code Listing 1 shows the registration of a feature detector. When a feature detector tries to register at the feature detector registry, the registry queries the information endpoint of the detector to determine its feature type. After that, the registry queries the feature detector quality determiner for the detector's quality. If the received quality is high enough, the registry informs the feature matcher by forwarding all relevant features. In contrast to ARTIFACT, the registration of the feature detector does not require the resulting pipelines, and the registry accepts the feature detector registration at this point.

```
void addDetector(InetSocketAddress address){
  FeatureDetectorInfo detectorInfo =
      ↪ requestDetectorInfo(address);
  FeatureDetectorQuality quality =
      ↪ requestDetectorQuality(detectorInfo
      ↪ , address);
  if (hasRequiredQuality(quality)) {
    Detector detector = new Detector(address,
        ↪ quality);
    addresses.put(detector.getName(),
        ↪ detector);

    List<String> allFeatures = addresses
      .values().stream()
      .map(Detector::getDetects).toList();
```

[4]https://spring.io/

```
    notifyFeatureMatcher(allFeatures);
  }
}
```

Code Listing 1: Registration of New Detectors.

6.3.4 Component Registry Extension

Compared to ARTIFACT, the component registry requires slight modifications to handle feature-enriched documents. Additionally, each extracted information now contains the quality of the used pipeline in order to determine if FAP is possible. Finally, as mentioned in Subsection 5.4, there is no longer one global pipeline for all documents. Instead, the system uses feature-based pipelines depending on the features of the received document. If the system cannot find an exact feature match, we use a pipeline whose features are closest to the features of the document. In this case, FAP is not possible.

6.3.5 Label Extender

The label extender microservice combines the label extension introduced in Subsection 5.1 and the pipeline generation into one single microservice. The label extender is a Python FastAPI web service that also replaces the pipeline quality determiner introduced in ARTIFACT. After label extension is performed, the service triggers the feature matcher service in order to generate document-specific pipelines.

Code Listing 2 illustrates the steps to perform label extension for all gold documents. For all information types, every pipeline is run against each document. If the pipeline result matches the expected result, the pipeline will be considered suitable for the document.

```
def extend_labels(self):
  extended_docs =[]
  for gold_doc in self.get_gold_documents():
    suitable_pipelines =[]
    for info in DomainModel.information:
      for pipeline in self.
          ↪ get_pipelines_for_information(
          ↪ info):
        result =PipelineExecutor.
            ↪ execute_pipeline(
          pipeline=pipeline,
          gold_document=gold_doc
        )
        if result ==gold_doc.get(info):
          suitable_pipelines.append(pipeline)
    extended_docs.append(
      LabeledGoldDocument(
        gold_document=gold_doc,
        suitable_pipelines=suitable_pipelines
      )
    )
  return extended_docs
```

Code Listing 2: Label Extension of Gold Documents.

6.3.6 Feature Matcher

As explained in Subsection 5.2, the feature matcher determines the best document-specific pipelines. As stated there, the current implementation features a rule-based matching strategy for pipeline generation.

Code Listing 3 illustrates the steps to create the feature-based pipelines. For each feature combination, we select all gold documents that match this exact feature set. After that, we extract the suitable pipelines we received from the label extender and calculate each pipeline's quality. This results in a list of all possible feature-based pipelines and their qualities. After feature matching is complete, the service sends all created feature-based pipelines to the component registry. This enables the registry to process incoming documents.

```
def match_features(self, labeled_docs: List[
    ↪ LabeledGoldDocument]):
  feature_based_pipeline_qualities =[]
  for feature_combination in self.
      ↪ generate_feature_combinations():
    suitable_pipelines =[]
    matching_gold_docs =self.
        ↪ match_labeled_documents(
        ↪ labeled_docs, feature_combination
        ↪ )
    for labeled_doc in matching_gold_docs:
      suitable_pipelines.extend(labeled_doc.
          ↪ suitable_pipelines)
    for suitable_pipeline in set(
        ↪ suitable_pipelines):
      feature_based_pipeline_qualities.append
          ↪ (
        FeatureBasedPipelineQuality(
          features=feature_combination,
          pipeline=suitable_pipeline,
          quality=suitable_pipelines.count(
              ↪ suitable_pipeline) /len(
              ↪ matching_gold_docs)
        )
      )
  return feature_based_pipeline_qualities
```

Code Listing 3: Feature-Based Pipeline Generation.

6.3.7 Document Processor

The document processor is a Java Spring Boot web service that provides a Camunda[5] process engine and extends the process orchestrator introduced in ARTIFACT. As stated in Subsection 5.5, this microservice

[5]https://camunda.com/

manages the overall process and decides if the information extraction quality justifies FAP.

When the user uploads a document for information extraction, the processor first calls the feature detector registry to extract the features of the given document. The resulting feature-enriched document is then forwarded to the component registry for information extraction. After extraction, the processor receives the extracted information together with their estimated quality. If the quality of the resulting information is not high enough (see Subsection 6.1), the extracted information must be checked manually. Otherwise, the information can be processed automatically.

Code Listing 4 shows the implementation of a Camunda service task that receives a document from the *DOCUMENT* variable and returns a document that additionally contains the detected features in the variable *ENRICHED_DOCUMENT*. This variable is then used in subsequent service tasks to query the component registry for the document information.

```
void execute(DelegateExecution exec) {
  FileValue docFile = exec.getVariableTyped("
      ↪ DOCUMENT");
  Document document = toDocument(docFile);

  FeatureEnrichedDocument serverResult =
      ↪ sendDocumentToServer(document);
  exec.setVariable("ENRICHED_DOCUMENT",
      ↪ serverResult);
}
```

Code Listing 4: Detection Service Task.

In Code Listing 5, the following Camunda service task receives the *ENRICHED_DOCUMENT* variable and requests the extraction of all information within the document. If the quality of the resulting information is not high enough, the extracted information must be checked manually. Otherwise, the information can be processed automatically.

```
void execute(DelegateExecution exec) {
  ObjectValue docFile = exec.getVariableTyped
      ↪ ("ENRICHED_DOCUMENT");
  FeatureEnrichedDocument document = docFile.
      ↪ getValue(FeatureEnrichedDocument.
      ↪ class);

  List<InformationWithQuality> result =
      ↪ sendDocumentToServer(document);
  if (result == null || result.isEmpty()) {
    throw new RuntimeException("No results
        ↪ received!");
  }

  double overallConfidence = result
    .stream()
    .mapToDouble(InformationWithQuality::
        ↪ confidence)
    .min().orElse(0);
  List<Information> information = result
    .stream()
    .map(InformationWithQuality::information)
    .toList();

  exec.setVariable("
      ↪ FULLY_AUTOMATED_PROCESSING",
      ↪ overallConfidence >= 0.95);
  exec.setVariable("RESULT", information);
}
```

Code Listing 5: Extraction Service Task.

7 CONCLUSION & FUTURE WORK

With FUSION, we provide an extension of ARTIFACT as an architectural pattern that enables the detection of document characteristics, called features, to generate document-specific IE pipelines.

ARTIFACT treats each input document the same and provides the automated generation of global-best pipelines. It also integrates a task for manual approvals, which is superfluous if an IE pipeline performs extraordinarily well for a specific document. Therefore, it does not enable FAP of documents.

To achieve document-specific processing and FAP, we introduced the concepts of feature matching (cf. Subsection 5.2), feature detection (cf. Subsection 5.3) and feature-based document processing (cf. Figure 13) to generate and use feature-based pipelines.

The experimental evaluation has shown that these concepts increase the pipeline results in comparison to the core concepts of ARTIFACT (cf. Table 4). Hence, the introduced approach supports the automated generation of IE without adding or improving existing pipeline components and, therefore, gains business value.

The evaluation also has shown that a simple rule-based feature matching strategy is suitable in the case of the introduced domain and relevant features. Furthermore, the introduced concepts empower the implemented system to find document groups for which specific IE pipelines perform extraordinarily well. Due to the automated feature detection of unknown input documents, the system automatically identifies the potential for FAP. In the case of the application in the real-world project, we identified a potential of 14% for FAP. Therefore, the concept of result routing (cf. Subsection 5.5) reduces the costs for superfluous manual approval tasks and gains business value.

We would like to apply different machine-learning-based algorithms for feature matching to find more suitable document classifications in future work. Also, we would like to use these algorithms to determine which features may have more or less impact on the optimal way of processing. In parallel, we would like to improve the quality of the detectors that currently do not reach the required quality criteria to consider the corresponding features.

REFERENCES

Aguirre, S. and Rodriguez, A. (2017). Automation of a business process using robotic process automation (RPA): A case study. In *Communications in Computer and Information Science*, Communications in computer and information science, pages 65–71. Springer International Publishing, Cham.

Albrecht, A. and Naumann, F. (2008). Managing ETL processes. In *NTII*.

Beck, K. (2003). *Extreme Programming - die revolutionäre Methode für Softwareentwicklung in kleinen Teams ; [das Manifest]*. Pearson Deutschland GmbH, München.

Borko, H. and Bernick, M. (1963). Automatic document classification. *J. ACM*, 10(2):151–162.

Cardie, C. (1997). Empirical Methods in Information Extraction. page 15.

Dumas, M., Van Der Aalst, W. M., and ter Hofstede, A. H. M. (2005). *Process-aware information systems*. John Wiley & Sons, Nashville, TN.

Evans, E. and Evans, E. J. (2004). *Domain-driven Design - Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, Boston.

Fowler, M. (2013). An appropriate use of metrics.

Hanson, C. and Sussman, G. J. (2021). *Software Design for Flexibility - How to Avoid Programming Yourself into a Corner*. MIT Press, Cambridge.

Hashmi, K. A., Liwicki, M., Stricker, D., Afzal, M. A., Afzal, M. A., and Afzal, M. Z. (2021). Current Status and Performance Analysis of Table Recognition in Document Images with Deep Neural Networks.

Ivančić, L., Suša Vugec, D., and Bosilj Vukšić, V. (2019). Robotic process automation: Systematic literature review. In *Business Process Management: Blockchain and Central and Eastern Europe Forum*, Lecture notes in business information processing, pages 280–295. Springer International Publishing, Cham.

Jablonski, S. and Bussler, C. (1996). *Workflow Management: Modeling Concepts, Architecture, and Implementation*.

Jiang, S., Pang, G., Wu, M., and Kuang, L. (2012). An improved k-nearest-neighbor algorithm for text categorization. *Expert Syst. Appl.*, 39(1):1503–1509.

Lilleberg, J., Zhu, Y., and Zhang, Y. (2015). Support vector machines and word2vec for text classification with semantic features. In *2015 IEEE 14th International Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC)*. IEEE.

Liu, Q. and Ng, P. A. (1996). Document classification and information extraction. In *Document Processing and Retrieval*, pages 97–145. Springer US, Boston, MA.

Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, first edition edition.

Rubin, K. S. (2012). *Essential Scrum - A Practical Guide to the Most Popular Agile Process*. Addison-Wesley Professional, Boston, 01. edition.

Saggion, H., Funk, A., Maynard, D., and Bontcheva, K. (2007). Ontology-Based information extraction for business intelligence. In *The Semantic Web*, Lecture notes in computer science, pages 843–856. Springer Berlin Heidelberg, Berlin, Heidelberg.

Schmidts, O., Kraft, B., Siebigteroth, I., and Zündorf, A. (2019). Schema matching with frequent changes on semi-structured input files: A machine learning approach on biological product data. In *Proceedings of the 21st International Conference on Enterprise Information Systems*. SCITEPRESS - Science and Technology Publications.

Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47.

Sildatke, M., Karwanni, H., Kraft, B., and Zündorf, A. (2022). ARTIFACT: Architecture for Automated Generation of Distributed Information Extraction Pipelines. In *Proceedings of the 24th International Conference on Enterprise Information Systems - Volume 2*, pages 17–28.

Skoutas, D. and Simitsis, A. (2006). Designing ETL processes using semantic web technologies. In *Proceedings of the 9th ACM international workshop on Data warehousing and OLAP - DOLAP '06*, New York, New York, USA. ACM Press.

Tarr, P., Ossher, H., Harrison, W., and Sutton, S. (1999). N degrees of separation: multi-dimensional separation of concerns. In *Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No.99CB37002)*, pages 107–119.

van der Aalst, W. M. P. (2004). Business process management demystified: A tutorial on models, systems and standards for workflow management. In *Lectures on Concurrency and Petri Nets*, Lecture notes in computer science, pages 1–65. Springer Berlin Heidelberg, Berlin, Heidelberg.

Wimalasuriya, D. C. and Dou, D. (2010). Ontology-based information extraction: An introduction and a survey of current approaches. *J. Inf. Sci.*, 36(3):306–323.