Integration of UIMA Text Mining Components into an Event-based Asynchronous Microservice Architecture

Sven Hodapp, Sumit Madan, Juliane Fluck, and Marc Zimmermann
Fraunhofer Institute for Algorithms and Scientific Computing (SCAI), Schloss Birlinghoven, Sankt Augustin, Germany
{sven.hodapp, sumit.madan, juliane.fluck, marc.zimmermann}@scai.fraunhofer.de

Abstract

Distributed compute resources are necessary for compute-intensive information extraction tasks processing large collections of heterogeneous documents (e.g. patents). For optimal usage of such resources, the breaking down of complex workflows and document sets into independent smaller units is required. The UIMA framework facilitates implementation of modular workflows, which represents an ideal structure for parallel processing. Although UIMA AS already includes parallel processing functionality, we tested two other approaches for distributed computing. First, we integrated UIMA workflows into the grid middleware UNICORE, which allows high performance distributed computing using control structures like loops or branching. While good distribution management and performance is a key requirement, portability, flexibility, interoperability, and easy usage are also desired features. Therefore, as an alternative, we deployed UIMA applications in a microservice architecture that supports all these aspects. We show that UIMA applications are well-suited to run in a microservice architecture while using an event-based asynchronous communication method. These applications communicate through a standardized STOMP message protocol via a message broker. Within this architecture, new applications can easily be integrated, portability is simple, and interoperability also with non-UIMA components is given. Markedly, a first test shows an increase of processing performance in comparison to the UNICORE-based HPC solution.

Keywords: UIMA, Microservice, Text Mining, Distributed Computing, Interoperability

1. Introduction

The Apache UIMA (Unstructured Information Management Architecture)[1] (Ferrucci and Lally, 2004) framework is one of the most used environments for the assembly of information extraction software. It defines standardized interfaces and allows multithreading. Multiple text mining modules are already integrated within UIMA. One example of a publicly available resource of UIMA components is DKPro Core (Eckart de Castilho and Gurevych, 2014). It provides a large collection of text mining modules wrapped within the Apache UIMA components using the uimaFIT library (Ogren and Bethard, 2009).

In addition to the availability of suitable text mining modules, distributed compute resources are necessary to extract information within large document collections such as full text papers or patents. UIMA Asynchronous Scaleout (UIMA AS)[2], which is part of the Apache UIMA project, allows distributed computing and can scale out UIMA applications using asynchronous messaging. It handles the messaging and the queue management necessary for inter-service communication using the open Java Message Service (JMS) industry standard. On top of UIMA AS, Distributed UIMA Cluster Computing (DUCC), extends its functionality towards distributed computing. It facilitates the scale out of UIMA and even non-UIMA applications and enables high throughput processing of large data collections. In addition, DUCC manages the life cycle of services deployed across a cluster.

In contrast to the afore mentioned work, we used the grid middleware UNICORE (Uniform Interfaces to Computing Resources) (Streit et al., 2010) to deploy and execute UIMA applications. For the compute intensive information extraction from large chemical patent collections, huge compute resources were necessary (Bergmann et al., 2012). The integrated text and image mining pipelines are based on UIMA and uimaFIT. In UNICORE, these applications are wrapped and deployed as *UNICORE GridBeans* to enable the distributed computing functionality. In addition, the Gridbeans contain the specification for input and output, needed compute resources as well as for configuration parameters of the application.

UNICORE offers a client and a server platform for grid computing and provides sophisticated workflow features as well as built-in application support. Deployed on a cluster system, it makes distributed computing possible in a seamless and secure way. Through a graphical user interface, the client software *UNICORE Rich Client* facilitates the setup of configurable workflows using control structures such as *loop*, *if*, *while* to combine different UIMA applications with other tools. Despite the high compute performance of UNICORE, new installations and configuration of GridBeans for users unaware of UNICORE is not seamless. In our experience maintenance and configuration of UNICORE is a complex task, and we assume a bottleneck in the massive usage of file I/O during stage in and stage out and in the service orchestrator for huge numbers of small jobs.

As a consequence, we searched for an alternative method. We tested the integration of UIMA into a distributed microservice architecture. In comparison to UIMA AS and DUCC, our microservices allow the design of event based systems that enable dynamic realizations of fine-grained text mining pipelines - we don't make use of predefined response queues or intelligent AS clients. In our case the message itself can contain the information where it should be routed next.

Many organizations such as Amazon, Google, Netflix have already evolved their platforms to microservice architecture

[1] https://uima.apache.org

[2] http://uima.apache.org/doc-uimaas-what.html

(Newman, 2015). They represent a new type of technology to tackle the challenge of rising complexity of an enterprise software system. The microservice architecture allows to break down a monolithic system into multiple components wrapped as small services. Decomposing a system in small services has various advantages, for instance faster delivery, embracing newer technologies, better scaling or easy deployment.

To improve interoperability, to ease deployment, and to accelerate large-scale processing, we integrated the UIMA components into a microservice architecture based on open source messaging broker Apache ActiveMQ Apollo[3]. In addition to the implementation details, we present a performance and scalability comparison with UNICORE grid computing and the microservice approach by applying a text mining workflow on a larger dataset. Furthermore, we analyze and discuss the findings and provide an outlook for future activities.

2. Material and Methods

First, we shortly describe the architecture of the UIMA Pipelets. They have been developed within the UIMA-HPC project[4] for the integration into UNICORE. The pipelet architecture allows the creation of modular information extraction workflows. For the integration into the microservice architecture, they were extended with several generic communication mechanisms. In the subsequent sections, further integration details of the microservice architecture are described.

2.1. UIMA Pipelet

The Pipelet Core Framework has been developed to integrate all kinds of applications into the UIMA ecosystem. We always bundle a reader (collection reader) and a writer (CAS consumer) with one or multiple annotators (analysis engines (AE)). A pipelet is basically a specialized aggregated analysis engine (AAE) helping the developer to easily build, configure, and deploy wrapped tools.

Figure 1 depicts the basic structure of a pipelet. In the following, the principal communication flow is sketched: first, the reader transforms the input data into a well defined CAS data structure. The main component of a pipelet–the analysis engine–takes the CAS information as an input, performs its annotation and/or extraction task, and enriches the CAS with the extracted structured information. The writer is able to transform the enriched CAS into the desired output format. Several readers and writers for plain text, PDF, CSV, SQL, DOCX, image formats, or SCAIView[5] are available. All pipelets use the serializable UIMA CAS data structure as the uniform exchange format. Also, a common type system specifies the needed data types, which is shared by all pipelets to handle the CAS data structure and providing provenance information. Further details of our UIMA pipelet architecture are described in Bergmann et al. (2012).

[3]http://activemq.apache.org/

[4]http://www.uima-hpc.de/en/about-uima-hpc.html

[5]http://scaiview.com

The Pipelet Core Framework includes the base libraries (e.g. UIMA, uimaFIT), utilities (e.g. provenance, parameter validation), and several generic communication mechanisms, which can be used by the pipelets as readers and writers. There are three different types of communication mechanisms available:

1. The type *File I/O* can read and write files directly from a file system,
2. *Pipe I/O* allows pipelets to read and write data streams from a UNIX pipe, and
3. *Message I/O* is used by the pipelets to exchange data as messages in the microservice architecture.

For the microservices, the last communication mechanism was newly included. Its implementation is detailed in the next section.

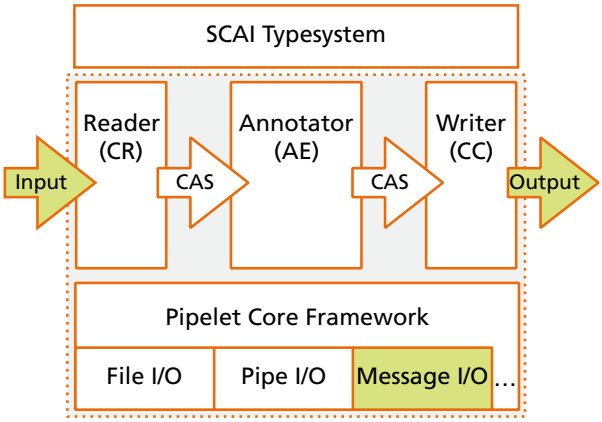

Figure 1: The basic structure of an UIMA pipelet is an aggregated analysis engine (AAE). Our microservice architecture can be addressed via Message I/O.

2.2. Pipelet as a Microservice

The implementation of the communication mechanism Message I/O in the Pipelet Core Framework allows us to deploy and execute each of our pipelets directly as a microservice. The implementation also provides capabilities to connect to a broker, maintain the connection, publish and subscribe to queues, and handles the messages.

2.2.1. Communication Method

The communication method describes how services communicate with each other. There are two major communication methods available: *request/response*, with which a client initiates a request and waits for a response; and *event-based*, which triggers the activation of services on incoming events. The request/response method is mostly used and implemented for synchronous tasks such as for web services or remote procedure calls. In contrast, the event-based method is preferred for asynchronous tasks, which doesn't require a response directly. Classically event-based systems are highly decoupled. Most of the UIMA components are independent by nature and are well suited for

the integration into a highly decoupled system. Therefore, we prefer the asynchronous event-based communication method.

2.2.2. Message Protocol

To enable a microservice architecture, it is important to have a common messaging protocol that is used to exchange data between microservices. Apache ActiveMQ Apollo supports various messaging protocols. From those, the Simple Text Orientated Messaging Protocol (STOMP)[6], is a lightweight and easy to implement protocol. It's design is similar to the popular and widespread Hyper Text Transfer Protocol (HTTP). Additionally, STOMP can be bridged to the Java Message Service (JMS) industry standard, which allows STOMP-based microservices to communicate directly with JMS-based applications.

Header	
content-type	gzip-xml
tracking-nr	Neoplasms-KW48
timestamp	1458661156661
event	ner.genes,store
agent	JProMiner (7.0) 28510@node-042
unit	13 [concepts]
license	MDAyOGxvY2F0aW9uIHNjYWl2a...
Body	
H4sIAAAAAAAAALzdXa+kx3Wm6fP5FQWeN5nr...	

Table 1: The general structure of a STOMP message. A set of key-value pairs builds the message header and the body contains the serialized CAS.

The STOMP-based messages are basically structured in two parts: A set of key-values as header entries and the message body (cf. Table 1). In case of our UIMA pipelets, the message body is simply an (compressed) XCAS. Every message requires the header property *destination* and may include *content-length*, *content-type* as additional properties. Those are part of the STOMP specification. For our text mining workflows, we introduce the following additional header properties:

- *tracking-nr*: For identification of related messages. For instance, all messages of the same document collection get the same tracking-nr.
- *timestamp*: This field contains the UNIX timestamp of the incoming message.
- *event*: It defines a vector of tasks. So in a workflow scenario each service knows where to route the message next, e.g. ner.genes, ner.chemicals, storage.
- *agent*: Contains information of the message sender, such as the program name and the machine (provenance).
- *unit*: In this property, every service can log information needed for accounting and service-level agreements (SLAs). Possible currencies might be the document length, the used CPU time, the license costs of the analysis engine, or the number of annotations.
- *license*: License information for process authorization is included in this property. For this, macaroons defined by Birgisson et al. (2014) are used. A macaroon is similar to a browser cookie, but in difference, it provides cryptographic signed caveats. This caveats can be checked decentralized by every involved microservice.

2.2.3. Message Broker

The communication is handled by the fast and reliable multi-protocol Apache ActiveMQ Apollo[7] messaging broker. It supports reliable messaging by persisting the messages in case of system failure. The persisted messages can be recovered and processed later. The broker represents the central well-known contact for all microservices. All microservice communication flows through the message broker.

For the delivery of a message, Apollo provides several types of destinations such as *queues* and *topics*. A queue represents a persistent message channel, which holds messages until a subscribed service picks them up. In such a way, queues have a load balancing property. In contrast, topics are non-persistent channels that drop messages in case of non-existing subscriptions. Also, they send every message to all subscribed services. As consequence, topics have a broadcasting property. Services can publish and subscribe to queues or to topics.

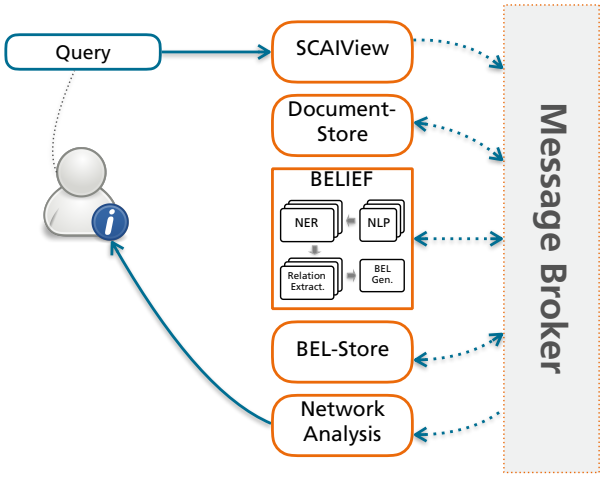

Figure 2: Illustration of a message flow of different microservices communicating with each other over a reliable message broker. Only the BELIEF components are UIMA pipelets.

2.2.4. Management

To manage the microservices, we introduced a management topic channel where all our services subscribe to. In addition, a library is integrated within the pipelets that includes management capabilities for the microservices. Based on

[6] https://stomp.github.io

[7] https://activemq.apache.org/apollo/

this library, it is possible to get the configuration settings, accounting logs, and statistic information. Furthermore, it is also possible to let the service unsubscribe and shutdown itself for maintenance. A graphical user interface has been developed which allows to configure, start, and monitor complex workflows (cf. Figure 3). All microservices have been registered to *Monit*[8], a flexible Unix toolbox for managing and monitoring Unix services. If a service fails to answer within 60 seconds it will be automatically restarted.

2.3. Workflow description

The event-driven asynchronous communication allows the definition and creation of flexible workflows. The workflow definition can be attached to each individual message as an event vector. The events specify which kind of services should be visited, therefore a per-message workflow can be defined.

A very complex retrieval and analysis task is to identify causal biomedical relationships within a set of articles. For instance, a researcher wants to know which drugs have an effect on different targets leading to a biological process in a certain disease context. For such a task, the user queries SCAIView to retrieve all relevant articles in the disease context. The result is a list of PubMed article identifers (PMID). The articles are retrieved via the PMIDs from a document store. Each article is sent to the BELIEF (Biological Expression Language Information Extraction WorkFlow) (Fluck et al., 2014) workflow, which itself is a collection of UIMA components communicating via messages. All extracted relationships are written back as BEL (Biological Expression Language)[9] documents into a BEL store. From the BEL store a cause-relationship network is generated and transferred into the Neo4j[10] graph database where all relevant paths are computed and presented to the user for inspection. The message communication flow is illustrated in Figure 2. The SCAIView client initiates a workflow task by sending a query and workflow plan to the document store over the message broker. All the services are listening to a queue for input and are sending the results to the next queue defined in the workflow plan, which is part of the message.

3. Results and Discussion

Both systems, the UNICORE as well as the microservice embedded UIMA workflow have been deployed to compare the performance and scalability. For the performance tests, 10 compute nodes with 16 cores each, 32 GB of RAM, and 56 GBit/s networking[11] were used. For each solution, one additional node was employed to host the UNICORE gateway and the ApolloMQ Apollo message broker respectively. All microservices have been queued on the compute cluster using the TORQUE Resource Manager[12].

We used a simple workflow that recognizes gene and protein names in text for our performance tests.

[8] https://mmonit.com/monit/#home

[9] http://www.openbel.org/

[10] http://neo4j.com

[11] Mellanox Infiniband FDR (56 GBit)

[12] http://www.adaptivecomputing.com/products/open-source/torque/

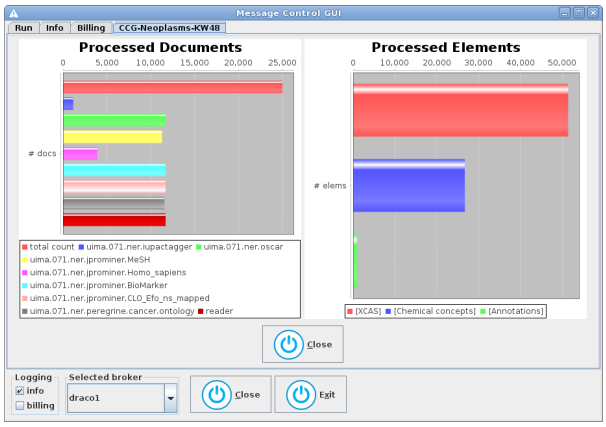

Figure 3: The graphical user frontend which allows to monitor microservices, configure, and launch workflows.

1. A SQL database is queried to retrieve a sample of one million PubMed abstracts. The SQL service creates a CAS for each document. In the message scenario, it generates a STOMP message and sends it to the gene annotator queue. In the UNICORE scenario, it creates an XMI file for each document and transfers it to the gene annotator grid bean.

2. ProMiner (Hanisch et al., 2004), the gene and protein UIMA annotator microservice, which is subscribed to this queue, gets the message, annotates gene information into the CAS, and sends the result to the destination queue. ProMiner grid bean gets the XMI files, annotates gene information into the CAS, and transfers the resulting XMI to the UNICORE storage.

Table 2 shows the results of the experiment. Even though both approaches used an equal number of processing nodes, microservices needed less overall processing time compared to the UNICORE-based approach.

Approach	Abstracts $[count]$	Time $[s]$	Performance $[abstracts/s/node]$
Microservices	$10k$	11	90.9
Microservices	$100k$	77	129.9
Microservices	$1M$	867	115.3
UNICORE	$10k$	102	9.8
UNICORE	$100k$	210	47.6
UNICORE	$1M$	1790	55.9

Table 2: Performance and scalability comparison of UNICORE and microservice approach using 10 cluster nodes.

Equally important is the ability to set up flexible workflows. With microservices, asynchronous, real-time or per-message text mining workflows can be build easily. Incoming new messages are queued and load balanced between all listening services. A uniform distribution of the message balancing could be observed during our tests. This is important since in general the documents to be processed are of different length, e.g. patents range from 1 to 500

pages. The documents are of different complexity, e.g. the number of chemicals extracted can vary from none to ten thousands for a single patent. And the documents are of different content, i.e. not all documents contain depictions or tables and some of them have passages in different languages. Therefore it is really hard to package and schedule jobs of same size for a set of diverse documents. Moreover, it is possible to absorb load peaks simply by starting the relevant microservices (temporarily) on our compute cluster. Another advantage of the microservice architecture is the inter-exchange between UIMA and non-UIMA services. Non-UIMA services can communicate over the same broker without interfering with the UIMA services in any way.

Other systems, such as the UNICORE solution as well as the UIMA AS solution, execute static pipeline plans. For every change in a pipeline plan, new aggregated workflows have to be assembled and deployed. In contrast, registered microservices are always available and allow to create flexible and even per-message grained workflows. In addition, fast response times can be expected. The scalability for batch processing can easily be reached through parallel deployment of the same services on multiple cluster nodes. Currently, those additional servers are started manually but in future, we plan to start and shut down these services automatically. Such automatic adaptation capabilities are also necessary to adjust systems with different analysis engines. Depending on the task, they can have very different performance characteristics.

The costs of integrating UIMA within the microservice architecture are rather low. No changes in the fundamental UIMA structure are necessary and we could use the communication abstractions of the Pipelet Core Framework. Therefore, it was easily possible to derive a first working version of the system. Moreover, now, those pipelines can be used in the UIMA framework alone, within the UNICORE and in the microservice environment without further changes.

On the management level, the inclusion of multiple services and distributed computing makes monitoring on different levels critical for the sustainability and success of the system. On the deployment side, automatic deployment and testing of new versions are necessary. For the monitoring of the services, all our services are subscribed to a management channel. In an asynchronous environment, services are built to make autonomous decisions (choreography pattern). Automatic throughput adjustments through starting and shutting down of additional services as mentioned above is a first future step in this direction. Moreover, we would like to develop self-organized workflows to make the per-message workflows more autonomous. For example, if more than two gene annotations are found in a text, the annotator might decide to pass the message to a relation extraction service. Such self-organization would save compute time considerably and would make configuration of workflows easier.

Mircoservices are well-suited to employ UIMA workflows in a distributed environment. The integration costs are low and the resulting services demonstrate a high degree of flexibility, interoperability, and scalability.

4. Acknowledgments

This work was supported by grants from the German Federal Ministry for Education and Research (BMBF) within the BioPharma initiative “Neuroallianz”, project I2 B ’Central IT Platform and RDF ProMiner Enhancement’ (grant number: 16GW0016), and by UCB Pharma GmbH (Monheim, Germany).

5. Bibliographical References

Bergmann, S., Romberg, M., Klenner, A., and Zimmermann, M. (2012). Information extraction from chemical patents. *Computer Science*, 13(2):21.

Birgisson, A., Politz, J. G., Taly, A., Vrable, M., and Lentczner, M. (2014). Macaroons : Cookies with contextual caveats for decentralized authorization in the cloud. *Ndss*, (February):23–26.

Eckart de Castilho, R. and Gurevych, I. (2014). A broad-coverage collection of portable nlp components for building shareable analysis pipelines. *Proceedings of the Workshop on Open Infrastructures and Analysis Frameworks for HLT*, 2(1):1–11.

Ferrucci, D. and Lally, A. (2004). Uima: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3-4):327–348.

Fluck, J., Madan, S., Ansari, S., Szostak, J., Hoeng, J., Zimmermann, M., Hofmann-Apitius, M., and Peitsch, M. C. (2014). Belief - a semiautomatic workflow for bel network creation. *Proceedings of the 6th International Symposium on Semantic Mining in Biomedicine (SMBM)*, pages 109–113.

Hanisch, D., Fundel, K., Mevissen, H.-t., Zimmer, R., and Fluck, J. (2004). Prominer : Organism-specific protein name detection using approximate string matching the prominer system. *BioCreative: Critical Assessment for Information Extraction in Biology*, pages 1–5.

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

Ogren, P. V. and Bethard, S. J. (2009). Building test suites for uima components. *Proceedings of the Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing (SETQA-NLP 2009)*, Proceeding(June):1–4.

Streit, A., Bala, P., Beck-Ratzka, A., Benedyczak, K., Bergmann, S., Breu, R., Daivandy, J. M., Demuth, B., Eifer, A., Giesler, A., Hagemeier, B., Holl, S., Huber, V., Lamla, N., Mallmann, D., Memon, A. S., Memon, M. S., Rambadt, M., Riedel, M., Romberg, M., Schuller, B., Schlauch, T., Schreiber, A., Soddemann, T., and Ziegler, W. (2010). Unicore 6 — recent and future advancements. *annals of telecommunications - annales des télécommunications*, 65(11-12):757–762, 12.