Authoring Processing Chains for Stream-based Internet Information Retrieval Systems

P. Katz[1], M. Feldmann[1], T. Lunze[2], S. Sprenger[1], and A. Schill[1]

[1] Technische Universität Dresden, Fakultät Informatik, Deutschland,
philipp.katz@tu-dresden.de
[2] Communardo Software GmbH, Dresden, Deutschland,
torsten.lunze@communardo.de

Abstract. Nowadays, Web-based information systems, such as web feeds or enterprise microblogs produce a seemingly continuous and endless stream of messages. Unfortunately, especially information workers currently experience an information overload. Thus, system support is required that enables a reduction of information load based on an automatic preprocessing of these streams. This paper presents an innovative approach to author IIR (Internet Information Retrieval) processing chains applicable for these stream-based business information systems. It is based on a novel system architecture named Spectre for realising highly scalable systems. Using a dedicated authoring tool, concrete systems can be developed efficiently and adapted to specific requirements. The solution has been validated using a prototypical implementation within a concrete business information system.

Key words: Enterprise 2.0, System Architecture, Information Aggregation, Information Retrieval, Component Architecture, Stream Processing

1 Overall Background and Motivation

In 2007, for the first time ever, more information was generated in one year than had been produced in the entire previous five thousand years – the period since the invention of writing. [1]

The biggest part of this information is not published via books and stored in libraries but via the Web. Due to the rise of the Web 2.0, a change in the way of gathering and accessing information took place. It is a very widespread approach, that information is not fetched by users but users are *subscribed* to information sources. This approach is reflected by RSS and Atom web feeds, by microblogs such as Twitter and by activity streams offered e. g. by Facebook. During the last years, this trend has reached the external as well as internal communication means of companies. Nowadays, companies offer various news and social media streams to communicate to customers or deploy internal systems such as enterprise microblogging systems, instant messaging solutions or Wikis to increase the transparency and efficiency of internal communication. As a result, many employees in such companies have to be subscribed to a variety of

information sources generating a potentially huge amount of information. This situation leads to a predicament: On one hand, a specific amount of information is highly relevant for their daily work. On the other hand, in many cases it is impossible to follow all the information flowing to the user.

It is obvious, that the problem of information overload is ubiquitous – covering the private as well as and especially the business related usage of current communication means. Thus, system support is absolutely essential to weaken the resulting problem of missing important information. A desired system should process the information automatically and present it in an efficiently accessible manner e. g. by grouping and ranking information or by proposing highly relevant information. Therefore, messages incoming from various stream-based information sources have to be routed through processing chains. A processing chain can be seen as a workflow description with specific IIR preprocessing and processing steps such as tokenization, token filtering, frequency calculation, part of speech (POS) tagging, named entity recognition (NER), text classification, keyword extraction, etc. As a result of our research and development efforts, we have created a novel approach simplifying the overall task from definition to deployment of such processing chains. This paper intends to provide a detailed overview of this approach.

The remainder of this paper is structured as follows. Section 2 discusses a use case to clarify the application area and gives an outline of our approach. In Section 3, an overview of the state-of-the-art is provided. In a further step, Section 4 introduces a highly scalable component-based platform for realizing IIR systems and discusses its characteristics in detail. Based on the information provided in Section 4, the novel authoring approach for developing stream-based IIR systems is discussed in Section 5. Section 6 concludes the paper and gives an outlook on future work.

2 Use Case and Approach Overview

The PRISMA information system has already been introduced in [2] and should be familiar to the reader, in order to get a better understanding of the following content. PRISMA forms a novel approach for enterprise communication. It allows its users to subscribe to various heterogeneous information sources as described in Section 1. To help the users to cope with the massive amount of incoming information, data gathered from these sources need to be processed by various IIR tasks that help to rank, group and interconnect gathered data. The illustrated steps take place in Phases 1 and 2 of the processing chain depicted in Figure 1. The system part carrying out the aggregation and preprocessing of incoming messages is named the system's *backend* while the part realizing the user interaction and user management is named *frontend*. The frontend interacts by sending subscription requests to the backend via a well-defined interface.

While the use case presented in [2] describes PRISMA from a end user's perspective interacting with PRISMA's frontend (see Figure 2), the following use case will focus on the backend part, and thus the distributed infrastructure and

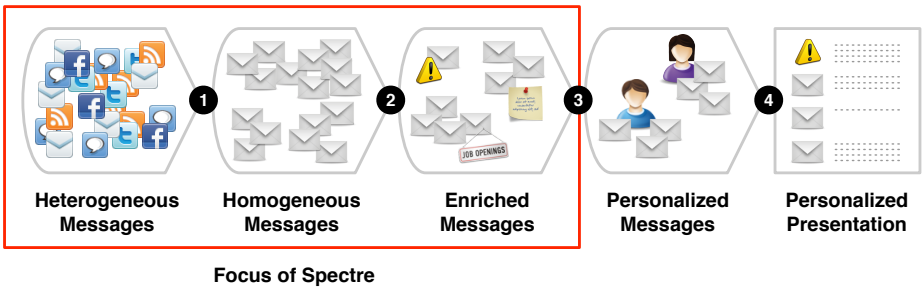

Fig. 1. Processing Chain (modified version of Figure 1 in [2])

the necessary processing steps for acquiring and processing information. Besides discussing the system architecture, this work will focus on the development methodology of such systems by IIR experts and developers as depicted in Figure 2. For this purpose an easy to use authoring tool is proposed which applies a dedicated authoring methodology discussed in Section 5. It empowers domain experts to specify IIR processing chains without programming skills using existing IIR components.

The following application scenario will be used to illustrate our work: ACME is a fictitious software company with 150 employees using PRISMA [2] to aggregate all relevant information streams. The individual knowledge workers at ACME use the system for consolidating and filtering information from various sources, providing them with a personalised information stream in the system's frontend with relevant information tailored to their individual interests. It is evident that the computation intensive IIR tasks applied to the messages gathered from the information sources result in problems in regards to the system's scalability.

After the processing chain of IIR tasks used by a system such as the mentioned one has been defined by IIR experts, it has to be implemented, configured and deployed. As we have made the experience during the development of various IIR systems, such as the mentioned prototype of the PRISMA research project, composing and setting up such a processing chain is highly time consuming and needs a very detailed understanding not only of the IIR domain but especially of building scalable software systems. Besides the huge effort that has to be taken for an initial development of an IIR processing chain, modifications and updates after deploying such a chain are currently labour-intensive. Due to these issues, we developed an overall solution to create and run processing chains within stream-based IIR systems, which consists of two core contributions:

1. A component-based software framework named **Spectre** used to run the processing chain in the form of a highly-scalable distributed system (discussed in Section 4).
2. An authoring tool named **Spectre Cockpit** used to define the processing chains, to configure the applied IIR components and to deploy the chain using the mentioned software framework (discussed in Section 5).

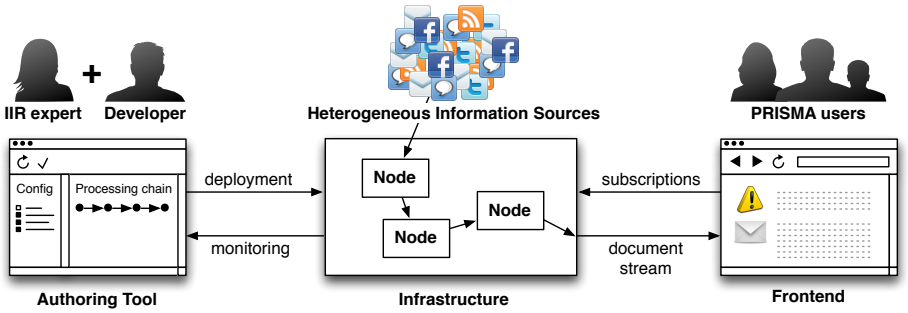

Fig. 2. Overview Spectre Infrastructure

The overall approach is depicted in Figure 2. Using the lightweight framework Spectre, the different IIR tasks are encapsulated into software components which are distributed to a set of available *Nodes*, each of them constituting a virtual or physical machine. The interaction between these components is realized by the Spectre framework as discussed in Section 4. The information determining the execution order of messages by the logic embedded into components and the configuration options for the embedded IIR logic is provided during deployment time by the authoring tool. This information can be updated at any time after the deployment of the components took place. Furthermore, the set of available IIR tasks can be extended, for example when new requirements to the preprocessing results arise.

3 State of the Art

After providing an overview of our novel approach, existing systems and solutions for creating processing chains for information streams from various heterogeneous sources are presented. Furthermore, graphical authoring tools for composing information processing and extraction workflows will be presented. Although, NLP toolkits and libraries such as OpenNLP or LingPipe provide basic building blocks, our considerations will focus on architectural aspects enabling a flexible composition and modelling of IIR workflows.

KNIME Desktop is an Eclipse-based platform which offers a component-based workbench for connecting different processing components using a so called pipes and filters approach. It can be used for information integration, data mining and analysis, and machine learning. Though its origins are in the area of bioinformatics, it can also be applied for exploring and experimenting with various data mining tasks in the field of IIR. KNIME Server is a solution to deploy and integrate such KNIME workflows within an existing company's infrastructure by providing the ability to access workflows using SOA paradigms [3]. A similar apporach is RapidMiner[1].

[1] `http://rapid-i.com/content/view/181/190/`

UIMA (Unstructured Information Management Architecture)[2] provides an architecture for processing, analyzing and extracting information from unstructured data sources such as text, audio, video and images. UIMA-based systems consist of a number of single components, which can be combined to an analysis workflow. Each component provides an isolated, independent task in the workflow to have a clear separation of concerns, thus allowing a strong modularity, recombinability and reusability [4]. UIMA uses a pipeline concept, where each component acts as an Annotator enriching segments in the processed information with specific metadata. UIMA AS (Asynchronous Scaleout) is an architectural approach for achieving scalability for UIMA workflows. Single analysis steps may be replicated and distributed among different logical or physical units. For exchanging data between components, a queue concept is being followed. Data between different nodes is exchanged using Apache ActiveMQ[3] JMS implementation, which provides rudimentary means for monitoring the individual queues' load by using JMX (Java Management Extensions).

AdaptIE [5] provides a language for Information Extraction (IE) tasks allowing to define extraction rules. AdaptIE provides a graphical user interface based on Eclipse focussing on two target groups, supplying each of them with an individual graphical interface: IE experts which are characterized by their fine grained knowledge in e. g. NLP and domain experts who contribute their domain-specific coarse-grained knowledge and build on the IE expert's knowledge.

The MapReduce [6] concept has gained significant attention over the last years, allowing parallel processing of so called big data. However, typical fields of application in the IIR domain are limited to tasks which can be well parallelized, like creating search indices for huge amounts of data, instead of establishing pipeline-based architectures.

None of the aforementioned approaches targets the complete lifecycle of setting up processing chains for IIR tasks. This starts with non-existent solutions which provide the ability to acquire data from various heterogeneous sources of different types of protocols and formats and integrating them into a common processing model with minimal manual effort. While the presented graphical tools such as KNIME Desktop or RapidMiner provide good means for initially prototyping concrete processing workflows, they lack the possibility of transferring such workflows to distributed and scalable infrastructures. Approaches focussing onto the architectural aspect such as UIMA only target parts of the scalability problem, especially do they provide no graphical tool support to set up and extend processing pipelines. Although rudimentary mechanisms for controlling the whole system's load exist, UIMA lacks elaborate techniques for automatically distributing an IIR system's components and to provide dynamic scaling properties of running systems, e. g. by deploying new or additional components to an existing infrastructure if needed.

[2] `http://uima.apache.org/`
[3] `http://activemq.apache.org/`

4 System Architecture

In this section the architecture of the Spectre framework is discussed. This discussion builds the fundament for providing details about the authoring approach in Section 5. As mentioned in Section 2, the backend offers an interface which allows subscriptions to various types of information sources. Thus, a system frontend sends subscriptions to the backend and receives messages from the associated information source which are then processed and enriched with metadata by Spectre's components. In the following, the central characteristics of this scalable backend will be discussed.

The central idea of the architecture is the distribution of processing chains over various nodes. For every subscription (e. g. an RSS feed or a Twitter account), an associated processing chain is determined through which messages resulting from this subscription are routed. Thus, every subscription is associated with one route through available processing components – these routes are named *Channel* in the Spectre terminology. For creating, updating and deleting subscriptions, a centralized system component is responsible which additionally determines an optimal processing chain. With this architecture, the management is centralized while the message streaming is decentralized. Due to this, the architecture is lightweight and easily understandable while highly scalable. The component-based system architecture consists of four core entity types:

1. A set of **Nodes** used as execution environment for Spectre's components. Every node may be a physical or virtual machine hosting a container in which the Spectre components can be deployed.
2. The system entity named **Coordinator** is responsible for scheduling a processing chain for messages associated with a specific subscription. Furthermore, it serves as point of interaction with a system frontend, thus offering an appropriate interface for subscribing to and unsubscribing from different information sources. Besides these tasks it monitors the overall system state and realizes component replication and distribution to achieve scalability.
3. A set of **Processing Components** per node which offer different IIR-specific functionalities such as stemming, stopword removal, tagging or relation detection. Besides IIR algorithms, logic for accessing information sources is encapsulated in Processing Components, called **Adapter Components**. Spectre offers a predefined set of Adapter Components for example for RSS feeds or Twitter which can be extended easily using Spectre Cockpit.
4. One **Hub** per node that manages the interaction between the components deployed on the associated node in accordance to the schedule provided by the Coordinator. Furthermore, the Hub realizes the interaction with further nodes and transfers messages to these nodes using a queue concept.

Figure 3 and 4 visualize the interaction between these entity types within a concrete example. The overall interaction is divided into three phases. During Phase I, the setup phase, components are initially distributed to nodes, instantiated and registered at the local Hub (Steps 1—3). Furthermore, the available Hubs register at the centralized Coordinator, providing information about the

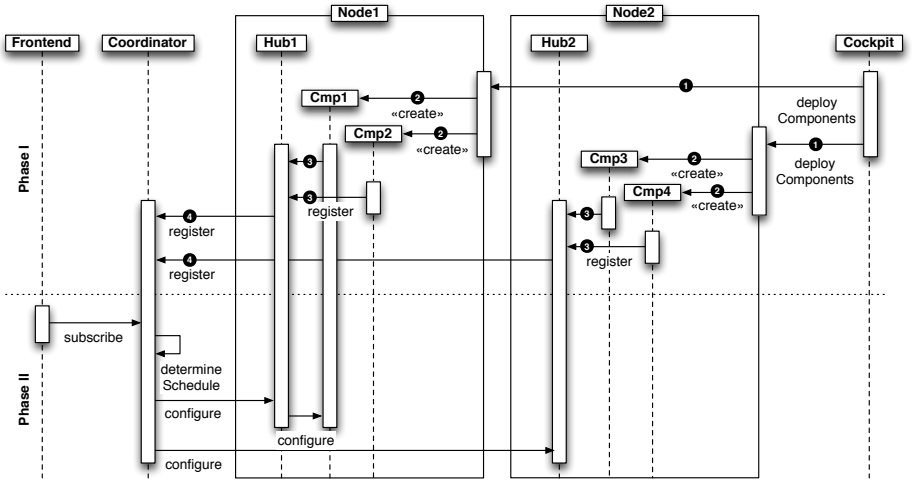

Fig. 3. Setup and Configuration Phase in Spectre

locally installed components (Step 4). Due to this procedure, every Hub can validate which components are available locally and the Coordinator can verify that all components and Hubs are correctly installed and can participate in processing chains.

After this setup phase, subscriptions can be accepted from a frontend in Phase II, the configuration phase. This phase is started by a frontend transmitting a subscription request to the Coordinator. Every subscription consists of an information source type, such as RSS or Twitter, and further source-specific data, such as authentication information. For every supported type, a specific Adapter Components is offered. It accesses the source and forwards the gathered information to the IIR processing chain. This chain needs to be specified for every subscription. Based on processing information specified by the system author, the processing steps needed for a particular source type are available within the Coordinator. These processing steps determine the order of the component types in the pipeline for each source.

Considering this processing chain, the Coordinator creates an optimal schedule based on the current system state and the capacity utilisation of different processing component instances of the necessary type. In regards to the steps depicted in Figure 4 it is e. g. determined, that incoming messages are processed by the components in the following order: `Cmp1`, `Cmp2`, `Cmp3` and `Cmp4`. Due to the components' distribution with ids `Cmp1` and `Cmp2` to `Node1` and with ids `Cmp3` and `Cmp4` to `Node2`, the determined schedule is communicated to the appropriate Hubs having the particular components under their local control. This configuration step is repeated for every subscription sent to the Coordinator.

After the configuration information has been distributed for one subscription, messages coming from the associated information source can be routed through the components. Every incoming message is processed by the respective component

and forwarded to the local Hub. Based on the processing chain description received by the Coordinator, the Hub forwards the message either to a further local component or transfers it to a remote Hub. In regards to the presented scenario, the message is forwarded to the Hub of `Node2` after it has been processed by component `Cmp2` so that it can be processed by `Cmp3` next. After having passed the whole processing chain, it is forwarded to the frontend.

During runtime, efficiency and capacity utilisation measures, like throughput and load of components are collected. These values are transferred to the local Hub in a first step. It analyses the collected values and in case, the Hub detects an overloaded component, the Coordinator takes care of rescheduling some of the established channels. The new schedules for the affected subscriptions are distributed to the infrastructure as described in Figure 3. Updates and modifications can be realized seamlessly. By extending or modifying the processing chain or component configuration using Spectre Cockpit, required changes can be transferred to the Coordinator and the participating nodes in the same manner as the initial setup (Phase I) described above. In case of extending processing flows, the additional components are made available ad-hoc to the infrastructure, the affected subscriptions are determined by the Coordinator and the changes are transmitted to the appropriate hubs. Thus, in sum, Spectre allows a transparent modification and extension of processing flows without stopping the processing and with no manual effort except the information provisioning via Spectre Cockpit.

The system architecture has been implemented and technically validated based on OSGi[4] as component execution environment using Apache Felix[5]. To simplify the development of the components, iPOJO[6] has been applied. Furthermore, XMPP[7] was used as a mechanism for inter-Node and Node-to-Coordinator communication. The different components implement a common Java interface which defines the API for controlling the components' lifecycles, for monitoring their workloads and for transferring messages.

5 Authoring Approach

After the Spectre framework used for deploying IIR processing chains has been discussed, the associated approach for authoring these chains is presented. As introduced in Section 2, the so called Spectre Cockpit provides an all-embracing authoring tool. It provides IIR experts with the possibility to create and adapt existing workflows when new requirements need to be considered. The authoring process thereby covers the following core areas:

1. New **Adapter Components** (see Section 4) can be created which are then used to acquire data from specific sources and integrate them into

[4] Open Services Gateway initiative
[5] `http://felix.apache.org/site/index.html`
[6] `http://felix.apache.org/site/apache-felix-ipojo.html`
[7] Extensible Messaging and Presence Protocol

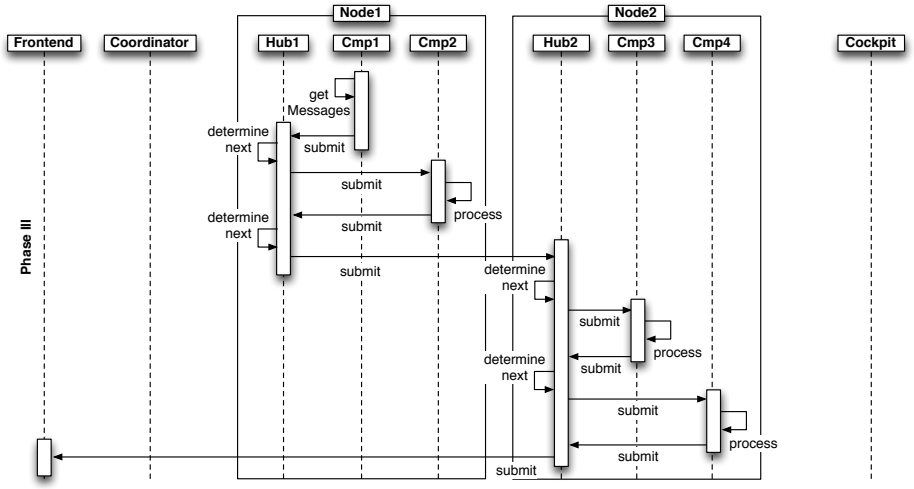

Fig. 4. Processing Phase in Spectre

Spectre's processing model. As described in Section 4, ready-to-use Adapter Components exist for a range of different source formats. When new types of sources need to be integrated, Spectre Cockpit supports experts with authoring new Adapter Components. This authoring process is realized fully visually without any need for programming. Using a data scheme or message instance from an external information source, the mapping of a source's data model to the Spectre data model is defined. Furthermore, protocol specific parameters are specified via Spectre Cockpit such as HTTP GET parameters. Based on this visually provided information, Adapter Components can be generated fully automatically.

2. New **Processing Components** for IIR tasks which perform subsequent steps after acquiring source data can be created. These usually carry out typical IIR specific tasks as outlined in Section 1. Similar to the described annotation concepts of state-of-the-art approaches (see Section 3), each Processing Component enriches the documents with specific metadata. Creating Processing Components for IIR tasks is the only step in the authoring methodology for which programming skills are required.
3. Existing components can be configured and combined to **Composite Components** and **Workflows**. Each single Processing Component performs a highly specialised and isolated task. Certain components might thereby require, that specific annotations are already supplied by the incoming document (e. g. a component performing a token filtering which requires an already tokenized document), other components depend on certain configuration parameters (like a language setting for a removal of stopwords). Spectre Cockpit provides the user with an intuitive, worflow-based interface for setting up processing chains and configuring each processing step.

4. **Testing** of components and workflows constitutes an important part of Spectre Cockpit. The authoring tool allows an incremental, experimental approach for creating sophisticated processing chains. Each component in the Workflow can be executed directly within the authoring tool to provide an efficient and error-free development methodology to the IIR expert.
5. After finishing the development process, Spectre Cockpit aids in the **Deployment** of the created workflows. The involved components are distributed to the available nodes and the necessary configuration options are distributed to the Spectre infrastructure.
6. Work on mechanisms for **Monitoring** workflows, nodes and their hosted components is currently in progress. Future versions of Spectre Cockpit will allow to monitor the measures of running systems and allow the user to control and influence strategies for replication and distribution of components.

The components and workflows defined by Spectre Cockpit are transferred to a specific repository from where they can be imported by further Spectre Cockpit instances or shipped to the Spectre execution infrastructure. The authoring tool has been implemented as Eclipse RCP (Rich Client Platform)[8] application, using Eclipse GEF (Graphical Editing Framework)[9] as foundation for the visual workflow editor. The current state of Spectre Cockpit is depicted in Figure 5.

To illustrate the potential of this tool, we build on the given scenario in Section 2. Imagine, CEO Bob decides, that ACME should use the existing PRISMA system to perform opinion mining tasks, monitoring various social media sources for positive and negative user feedback about ACME's products. Mary, responsible as an IT and IIR expert for administrating PRISMA, therefore initially needs to integrate new source types into the processing chain.

Both, Twitter and Facebook are currently not supported by the existing deployment, which means, that new Adapter Components need to be created by using Spectre Cockpit's facilities. Furthermore, Mary requires additional Processing Components for achieving her goal. To detect, which products are mentioned in the processed documents, Mary needs an NER, which is already available as component for Spectre. Additionally, Mary decides to use a text classification algorithm in order to detect the customers' sentiments in the documents. In the described example, such a component is not yet available. Therefore, Mary asks programmer Joe for assistance. Using the Spectre Cockpit, Joe is able to integrate a text classification implementation as new component into Spectre. From his code, a new component which can be integrated into Spectre's pipeline is generated.

After the necessary prerequisites have been fulfilled, Mary can start with creating a new workflow using Spectre Cockpit. It acquires data from Facebook and Twitter using the dedicated components. The incoming data is then piped through various Processing Components like a tokenizer, token remover, etc. After that, a NER is performed by the respective component to detect text

[8] `http://wiki.eclipse.org/index.php/Rich_Client_Platform`
[9] `http://www.eclipse.org/gef/`

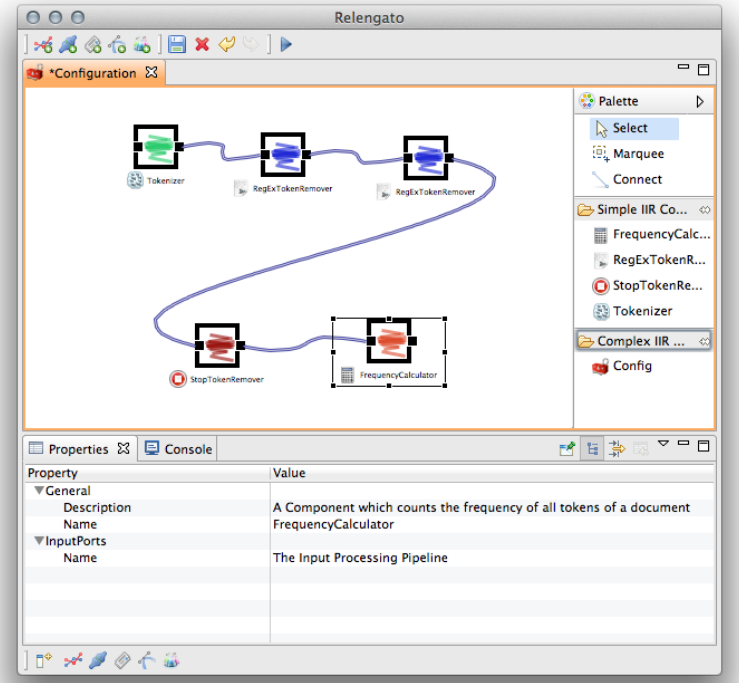

Fig. 5. Processing chain visualized in Spectre Cockpit

occurrences of ACME's products. Using the newly created text classification component, users' sentiments are detected for each document, assigning metadata to each document describing the sentiment detection's results.

To summarize, the result of the authoring process consists of the following integral parts: A set of specialized Processing Components performing specific tasks (like the NER and the text classification components in the given example), the configuration options of these components and a workflow description connecting the respective components. This workflow is finally deployed to an existing infrastructure. Spectre takes care of an ad-hoc deployment, distributing the involved components transparently to the available nodes as discussed in Section 4. With this we have provided a detailed overview of the Spectre authoring approach for defining processing chains for stream-based information systems which has been validated by a prototypical implementation applied to real world scenarios in the course of the PRISMA project.

6 Summary and Outlook

This paper introduced a novel approach for authoring and executing stream-based IIR systems. The system is based on an easy to apply component-based structure. The core idea is to distribute different IIR tasks using specialized components

over various nodes. Using the dedicated authoring tool Spectre Cockpit, concrete processing flows can be authored and distributed to an infrastructure consisting of various physically distributed nodes. Spectre Cockpit allows efficient modelling of processing workflows for IIR experts by resorting to preexisting Processing Components. Users are isolated from technical aspects, such as deployment and configuration of these components on a sophisticated, distributed architecture. Furthermore, IIR experts are isolated from implementation specific details by providing them an easy-to-use interface for settings up complex processing chains using a visual authoring tool. This paper presented the overall approach. We plan to focus on details of the single aspects and mechanisms used in our approach in several follow up papers.

In future work we will focus on enhancing the scalability of the system by improving the algorithms and means for IIR component distribution and replication. This includes especially the optimization and evaluation of the rescheduling algorithm mentioned in Section 4. Furthermore, an essential aspect in regards of the decentralization of messages will be addressed. The messages are streamed via different ways through the system. Some IIR algorithms need a global perspective on the available data (e. g. for learning models or corpora). It has to be analysed how data or core characteristics of this data can be spread via the components in an efficient manner in order to achieve this global perspective.

Besides these research related issues, it is planned to publish Spectre as an open source project until the third quarter of 2012.

Acknowledgments. The contributions presented in this paper are results of the research project PRISMA which is developed in a cooperation between the Technische Universität Dresden and the Communardo Software GmbH. The PRISMA project is funded by the Free State of Saxony and the EU (European Regional Development Fund).

References

1. J. Bloem, M. van Doorn, S. Duivestein. *Me the media – Rise of the Conversation Society*, VINT editions (research institute of Sogeti), 2009, p. 270.
2. P. Katz, T. Lunze, M. Feldmann, D. Röhrborn, A. Schill. *System Architecture for handling the Information Overload in Enterprise Information Aggregation Systems*, BIS 2011; Poznan, Poland; 6/2011.
3. M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, B. Wiswedel. *KNIME: The Konstanz Information Miner*, Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007).
4. D. Ferrucci, A. Lally. *UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment*, Natural Language Engineering, 2004.
5. W. M. Barczyñski, F. Foester, F. Brauer, D. Schuster. *AdaptIE – Using Domain Language concept to enable Domain Experts in Modeling of Information Extraction Plans*, 12th ICEIS; Funchal, Madeira, Portugal; 2010.
6. J. Dean, S. Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters*, OSDI'04; San Francisco, CA, 2004.