TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Petri Rantanen

Architecture for Interfacing Content Analysis Back Ends

Julkaisu 1460 • Publication 1460

Tampere 2017

Tampereen teknillinen yliopisto. Julkaisu 1460
Tampere University of Technology. Publication 1460

Petri Rantanen

Architecture for Interfacing Content Analysis Back Ends

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in Auditorium 125, at Tampere University of Technology – Pori, on the 24 of March 2017, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2017

ISBN 978-952-15-3912-1 (printed)
ISBN 978-952-15-3924-4 (PDF)
ISSN 1459-2045

Abstract

This thesis will discuss the challenges of designing and implementing a complex system for the content analysis ecosystem. The system would consist of any number of client devices – both desktop and mobile clients – and of a single front-end service, which would utilize any number of external service providers and back ends designed for various content analysis tasks. These tasks could include, among others, text summarization, photo analysis or content-based search. The background research suggests that despite the extensive studies on content analysis in general, there is a lack of research on how to implement a usable system consisting of all of the required parts. In existing studies the priority is almost entirely placed on algorithm and analysis method design.

The purpose of the research presented in this thesis is to study how to define a generic, extendable, and maintainable system within the content analysis domain. This thesis describes the methods, technologies, and principles required in architectural design. Furthermore, the work is validated through several architectural iterations and proof-of-concept implementations, which are also presented in this thesis.

Based on the results of the studies performed, this thesis will describe how to realize a feasible and practical generic architecture by following layered Application Programming Interface (API) and data models (presented in this thesis) in combination with commonly used architectural and technical solutions (Representational State Transfer (REST) and hybrid REST/Remote Procedure Calls (RPC)) and industry de facto representation formats (such as JavaScript Object Notation (JSON) and Extensible Markup Language (XML)).

Preface

The background of the research presented in this thesis, and my contribution to it, as well as the many projects I have participated in are covered in detail in the following chapters of this thesis. Thus, I would like to use this space to thank those who have helped me to realize this thesis.

I started as a researcher in the Pori Department of Tampere University of Technology in 2008 with the goal of finishing my Master's thesis. This also laid the foundations of the research presented in this doctoral thesis. The original topic of my research, as presented in my Master's thesis – that is, emergency messaging – did not carry through the years, but in essence the research was about software architecture and interface design. In addition to the research presented in this thesis, I have worked in several other projects – with topics such as the development of systems for eldercare, the measurement of environmental conditions in a hospital, the improvement of data collection in a swimming hall, and the design of a system for tracking logistics. All these topics, in one way or another, were closely related to architectural design – to making interfaces, which allowed systems to communicate with other systems. Ultimately, the context changed to content analysis, as presented in this thesis.

Thus, taking into account several years of research with many colleagues and co-authors, I would like to thank everyone who has worked with me, both at the university and in the companies participating in the research projects, for providing directions for my research, feedback and comments on my publications, and for making it possible to realize the required proof-of-concept implementations, prototypes and system specifications.

I would also like to thank Dr. Jari Soini and Prof. Hannu Jaakkola for providing guidance on the process of writing my thesis. Similarly, I would like to extend my gratitude to my thesis pre-examiners, professors Jaak Henno and Marjan Heričko, for providing valuable insight, feedback, and improvement ideas.

I would also like to express additional thanks to Tampere University of Technology, the Finnish Cultural Foundation Satakunta Regional Fund and Kaarina Vaalannon Rahasto for their financial support in making it possible for me to complete this thesis.

List of Figures

List of Tables

List of Abbreviations and Definitions

ADL	Architecture Description Language
API	Application Programming Interface
ATAM	Architecture Tradeoff Analysis Method
CAP	Common Alerting Protocol
CBIR	Content-Based Image Retrieval
CDN	Content Delivery Network
CRUD	Create, Read, Update and Delete
DMZ	Demilitarized zone
Exif	Exchangeable image file format
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
ISCRAM	Information Systems for Crisis Response and Management
JSON	JavaScript Object Notation
MMM	Mathematical Model of Meaning
NICT	National Institute of Information and Communications Technology
OS	Operating System
OSI	Open Systems Interconnection
PDF	Portable Document Format
REST	Representational State Transfer
RPC	Remote Procedure Calls
RSS	Rich Site Summary
SSL/TLS	Secure Sockets Layer / Transport Layer Security
SOA	Service-Oriented Architecture

SoC Separation of Concerns

TCP/IP Transmission Control Protocol / Internet Protocol

UI User Interface

UML Unified Modeling Language

URI Uniform Resource Identifier

URL Uniform Resource Locator

VPN Virtual Private Network

W3C World Wide Web Consortium

WSDL Web Services Description Language

XML Extensible Markup Language

XSLT Extensible Stylesheet Language Transformations

List of Publications

I Sillberg, P., **Rantanen P.**, Saari, M., Leppäniemi, J., Soini, J. and Jaakkola, H., "Towards an IP-Based Alert Message Delivery System", in Proceedings of the 6th International ISCRAM Conference, ISBN 978-91-633-4715-3, Gothenburg, Sweden, May 10-13, 2009.

II **Rantanen P.**, Sillberg, P., Jaakkola, H. and Nakanishi, T., "An Asynchronous Message-based Knowledge Communication in a Ubiquitous Environment", Database Systems for Advanced Applications, Springer, ISBN 978-3-642-14588-9, DOI 10.1007/978-3-642-14589-6_44, pp. 434-444, 2010.

III Sillberg, P., Kurabayashi, S., **Rantanen P.** and Yoshida, N., "A model of evaluation: computational performance and usability benchmarks on video stream context analysis", Information Modelling and Knowledge Bases XXIV, IOS Press (Frontiers in Artificial Intelligence and Applications; Volume 251), ISBN 978-1-61499-176-2, DOI 10.3233/978-1-61499-177-9-188, pp. 188-200, 2013.

IV **Rantanen P.**, Sillberg, P. and Soini, J., "Content Analysis System for Images", in Proceedings of the 16th International Multiconference Information Society, IS 2013, Volume A, 7-11, Ljubljana, Slovenia. Josef Stefan Institute, pp. 241-244, October 7-11, 2013.

V **Rantanen P.** and Sillberg, P., "Event Calendar for Internet Data Sources", in Proceedings of the 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, Croatian Society for Information and Communication Technology, DOI: 10.1109/MIPRO.2014.6859721, pp. 1035-1040, May 26-30, 2014.

VI **Rantanen P.**, "REST API Example Generation Using Javadoc", Computer Science and Information Systems, ComSIS Consortium, ISSN 1820-0214, 2017. *Accepted for publication.*

VII Ahmad, I., **Rantanen P.**, Sillberg, P., Laaksonen, J., Liu, S., Forss, T., Malik, A., Nieminen, M., Shetty, R., Ishikawa, S., Kallio, J., Saarinen, J. P., Gabbouj, M. and Soini, J., "VisualLabel: An Integrated Multimedia Content Management and Access Framework for Personal Users", in Proceedings of the 27th International Conference on Information Modelling and Knowledge Bases (EJC 2017), Krabi, Thailand, June 5-9, 2017. *Submitted.*

Author Contribution

The contribution of the author of this thesis for the included publications is listed here for the reader's convenience. The contribution of the publications in relation to the topic of this thesis is further explored in the chapter 3, and the chapter also includes brief summaries of the main content of the publications.

Towards an IP-Based Alert Message Delivery System

The studies presented in the article, Publication I, are an extension of the work first presented in the author's Master's thesis [Rantanen and Sillberg, 2009]. The primary task of the author included the design and implementation of the client – server communication, including programming the client application used for testing the system implementation. The author also studied the literature related to the alert messaging use case presented in the publication with the goal of finding the optimum communication protocols and data formats. The author of this thesis presented the work at the sixth International Information Systems for Crisis Response and Management (ISCRAM) conference in May 2009.

An Asynchronous Message-based Knowledge Communication in a Ubiquitous Environment

In Publication II an extension to the alert message delivery system [Publication I][Rantanen and Sillberg, 2009] is presented. The work was carried out as a co-operation project between Tampere University of Technology, Keio University Shonan-Fujisawa Campus (SFC) and National Institute of Information and Communications Technology (NICT). The extension was designed during the author's visit to Japan over a three-month- long research exchange period. The author participated in designing the communication protocols utilized in the proof-of-concept implementation and wrote the code for the test client application used to validate the system functionalities. The author was also involved in the design and implementation of the server-side end-points for client – server and server – back end communications.

A model of evaluation: computational performance and usability benchmarks on video stream context analysis

The author's contribution to Publication III includes the description of the overall architecture used to implement the video analysis service, and the design of the interfaces and request/response formats used in the communication between the web service and clients and between the web service and the video analysis back end. The author also participated in the design of the test scenarios and in the execution of the performance

benchmarks. The design and implementation work for the system was done during the author's visit to Japan over the three-month period. The video analysis algorithms and the back end implementation were provided by the Japanese co-authors of the paper. The work was presented by the co-author Pekka Sillberg at the 22nd European Japanese Conference on Information Modelling and Knowledge Bases (EJC 2012) in Prague, Czech Republic in June 2012, and published in the book Information Modelling and Knowledge Bases XXIV in 2013.

Content Analysis System for Images

Publication IV presents a concept architecture developed for managing digital data using content-based analysis targeting photographic content as an example use case. The primary contribution of the author to the publication is in the design of the task-based approach utilized for delivering analysis tasks (a method further discussed in this thesis). The author also co-designed the data formats specifically for the photo analysis use case and helped to form the integration specifications for interfacing with the external back ends utilized in the analysis system. The author was the principal author of the publication and presented the work at the 16th International Multiconference Information Society (IS 2013) in Ljubljana, Slovenia, in October 2013.

Event Calendar for Internet Data Sources

Publication V presents an event calendar service that can be used to combine data from multiple sources. The core platform is based (sharing the same codebase, and thus, is implemented on the same design principles) on the framework presented in Publication IV. The author designed the server-side interfaces for the presented service and formulated the overall architecture. The author was the principal author of the publication and presented the work at the 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO) in Opatija, Croatia, in May 2014.

REST API Example Generation Using Javadoc

Publication VI describes how source code based service declarations in combination with tools commonly used with the Java programming language can be used to automatically generate documentation and use case examples. The author was the sole author of the publication and formulated the concept of utilizing existing method responses for use case example generation, as presented in the paper. The architecture used to generate the examples is the same as previously described in Publication IV and Publication V. The partial API model presented in the paper (and further expanded in this thesis) is based on work done by the author.

VisualLabel: An Integrated Multimedia Content Management and Access Framework for Personal Users

Publication VII presents an integrated framework – called VisualLabel – for managing multimedia content by utilizing a front-end service in combination with multiple analysis back ends. The article presents a simplified version of the API model that is further explored in this thesis. Furthermore, the core VisualLabel framework introduced in the publication is based on the design principles presented in this thesis.

The article summarizes several years of co-operation between the industrial and educational partners on the Data to Intelligence (D2I) project. The main outcome of this work was

the creation of the VisualLabel framework. The core functionality of the front-end service (such as task scheduling, client interfaces, and external service connectivity) and the integration specification was designed and implemented – with active feedback from the other partners – at Tampere University of Technology, Pori Department. The author of this thesis worked for the entire duration of the project on the development of the VisualLabel framework, with the primary task of designing the integration specification to enable the realization of the framework. The crucial parts of the specification (service interfaces, task scheduling, and automatic documentation) are further discussed in this thesis. The author of this thesis was also the corresponding author for the publication.

Table of Contents

1 Introduction

The growth of the Internet and the availability of information and various multi-format data have created challenges for information processing and retrieval. This has made it more difficult to find the desired information. This is not a new phenomenon, but something that has been going on for at least the past decade – fueled by the increase in public awareness of the Internet. Modern, easy-to-use mobile devices and online services have increased user expectations. Simple keyword-based search services are being replaced by semantically associative search engines that depend heavily on metadata.

This raises the question of how to gather or generate this metadata? There are three commonly used methods for producing the required metadata:

1. Metadata is directly provided by the user in the form of manually inserted tags and annotations, for example, tag clouds on various image-hosting services such as Flickr or Picasa, or more recently on social media websites such as Facebook or Twitter.

2. Metadata is inserted automatically into the user's content during content creation or modification. An example of this is the inclusion of Exchangeable image file format (Exif) [Camera & Imaging Products Association, 2015] metadata into image files by image capturing software when the user takes pictures with his/her camera.

3. Automatic generation of metadata directly from the user's content (or files). The process can take advantage of the metadata created by one or both of the methods described above – for context recognition, for example.

The metadata provided by the user (method 1 in the list above) has two basic problems. Firstly, even though the data is inserted by the user and may as such be considered to be "correct," it is not necessarily refined enough to be used in complicated search operations. Secondly, and in this case, more importantly, manually annotating a huge amount of content can be a cumbersome, time-consuming, and tedious process from the user's point of view. Method 2 overcomes some of these problems by offering a more standardized format and lessens the user's burden in the generation process. Unfortunately, the metadata generated this way is often very limited, and may well answer the questions *where*, *when* and *how* the content was created, but provides very few answers regarding *what* the content actually is. Method 3 can be used to answer the last question, i.e., what the content is and what it contains, and in some cases it can also be used to provide answers for where, when, and how. In the scope of this thesis, method 3 is perhaps the most important, although a fully functioning content analysis system targeted for end users might need functionality for implementing all three methods.

In essence, these three questions and the world of metadata generation as a whole provide

the higher context for this thesis, but is in its entirety a far too complex and wide research field. Thus, the following sections will attempt to slice this world into smaller and more manageable entities and will explain the scope of this thesis and go more deeply into the research problem described – and solved – in this thesis.

1.1 Research Area

Content analysis [Krippendorff, 2012], according to Tipaldo [2014] (and translated by Wikipedia [2016]) consists of "a wide and heterogeneous set of manual or computer-assisted techniques for contextualized interpretations of documents produced by communication processes (any kind of text, written, iconic, multimedia, etc.) or signification processes (traces and artifacts), having as ultimate goal the production of valid and trustworthy inferences". In the context of this thesis, the scope of content analysis is more limited, and the thesis focuses more on the technologies and existing systems that produce content analysis services or functionalities. The technologies are explored through the study of programming and communication interfaces, data formats, and other methods that enable better interoperability of services, systems, and platforms.

There are many – both competing and complementing – algorithms, frameworks, and platforms for automatic metadata generation. These solutions can be specialized to handle one or more data formats or media types (text, audio, image, video) and the metadata provided by the solutions may or may not be interchangeable or compatible with each other, or the data may not even be usable outside the context of the analysis engine in question.

The interoperability issues between content analysis engines can be considered a more specialized – though equally challenging – case of a syntactic and semantic interoperability problem as found when interfacing traditional software components and systems. In the context of this thesis, the analysis engines (or back ends) are independent and possibly complex systems, which can be used to generate metadata on the provided content. The back ends provide their services by means of interfaces reachable over the Internet (for example, by using the client–server approach) or inside a more limited network structure (for example, in a demilitarized zone or in a cloud).

Despite excessive research on the topic of content analysis, and on technology in particular, there is still a limited number of practical implementations. One of the problems is that most of the existing research focuses more on the design of the Content-Based Image Retrieval (CBIR) systems themselves [Antonelli et al., 2006; Lee and Guan, 2004; Rahman et al., 2007; Trojacanec et al., 2009][Publication VII], on distributing the CBIR system [Lee and Guan, 2004; Müller et al., 2003; Robles et al., 2005], or on the metrics or general principles of the analysis benchmarking [Datta et al., 2008; Kosch and Maier, 2010; Vogel and Schiele, 2006] – all of which are important topics themselves – without factoring in the larger architecture required to implement a usable system. Additionally, in many cases the various CBIR implementations can be considered to be competing with each other, even though a more practical approach would be to combine the systems under a single extendable generic architecture.

Similar observations can be made for text [Gupta and Lehal, 2009; Liang et al., 2005; Maramba et al., 2015; Ntoulas et al., 2006; Salton, 1968; Tsytsarau and Palpanas, 2011; Yoshitaka and Ichikawa, 1999], audio [Fu et al., 2011; Lu, 2001; Schedl et al., 2014; Typke et al., 2005; Yoshitaka and Ichikawa, 1999] and video analysis [Hu et al., 2011; Junga et al., 2004; Money and Agius, 2008; Swanberg et al., 1992; Yoshitaka and Ichikawa, 1999],

or on combining multiple features of a single media source [Atrey et al., 2010] (such as analysis of the video file's audio and image track): the research spans many fields and various topics, but the studies on the supporting – or enabling – architecture seem to be lacking. Of the four multimedia types (audio, image, video, and text), perhaps text – in the form of various Internet search engines – and audio have seen the most progress in commercial end-user products. Also, a certain level of video processing takes place on popular video sites (such as YouTube, Vimeo) and in online stores (iTunes, etc.). There has also been research on multi-modal fusion – the combination of different types of features from a single media source, such as combining the analysis of a video's audio and video track.

At this point it is important to place another limit on the scope of this thesis. The research presented in the thesis does not go very deeply into the inner workings of the content analysis systems. This is partly because of the existing extensive research already performed by others, and partly because the aim of the study is not to limit the content analysis systems to any particular type of analysis, such as text or video analysis, but to consider how these different types of analysis systems (or platforms) could be unified and utilized in a more general fashion. This is also reflected in the choice of various types of back ends used to validate the generic architecture presented in this work.

Like content analysis, the other important topic of this thesis, service API design, has been an active research topic, and there are numerous publications on REST architecture [Belwasmi et al., 2011; Fielding, 2000; Fielding and Taylor, 2002; Pautasso et al., 2008; Richardson and Ruby, 2007] and RPC [Birrell and Nelson, 1984; IETF, 1976; JSON-RPC Working Group, 2015; W3C, 2007a]. Additionally, if one considers the World Wide Web Consortium (W3C)'s definition of "Web Service" [W3C, 2004b], the languages and solutions used to describe the service interfaces – such as Web Services Description Language (WSDL) [W3C, 2007b] – should also be included. Similarly, there have been a lot of studies of content-based analysis [Datta et al., 2008; Hanbury, 2008; Lew et al., 2006; Zhang et al., 2012], metadata generation [Cardinaels et al., 2005], and metadata interoperability [Haslhofer and Klas, 2010].

Evidently, there have been a lot of studies on software architecture on a more general level, and even though the researched architectures share similarities with content analysis systems, there does not seem to be a comprehensive study on how the architectures should be used within the content analysis domain. The existing content analysis research prioritizes the algorithm and analysis methods with less weight given to the actual software solutions required to implement a viable system (or analysis environment), despite both being equally important research topics. Finally, there are undoubtedly real-life implementations, which adhere to similar principles as those presented in this thesis, but these implementations are often closely guarded by companies and not available to the public. Furthermore, it is difficult to deduce how generic the implementations are without access to the specifications.

In any case, the problem can be simplified to a case in which there is content (text, video, audio, etc.), which has been produced either by the clients (users) or by utilizing some kind of automatic process. Depending on the content type, different methods (algorithms) to produce the desired outcomes must be chosen, and because of the multitude of content types it is unlikely that a single solution or system can handle everything. For this reason, multiple analysis engines (back ends) and a method of delivering the content to these engines are required. In the context of this thesis, the deliveries that contain the necessary information and possible analysis instructions are called tasks. The task in combination

with a simplified use case including the essential participants of the content analysis environment is presented in Figure 1.1.

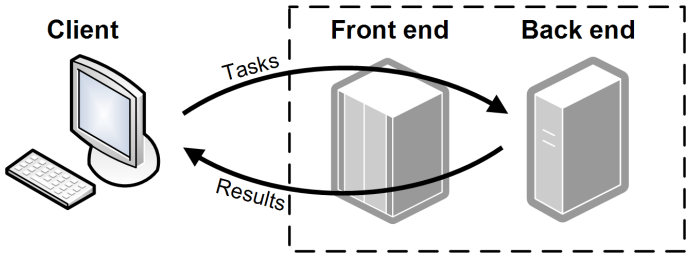

Figure 1.1: The simplified use case.

The clients are not strictly defined and may be anything from online photo browsers to native navigation applications – all depending on the type of metadata generated by the back end. A back end that crawls the Internet for event information could enable a calendar application [Publication V], which lists the data in chronological order. Another back end could analyze photos and extract metadata describing the contents of the photo, thus enabling an online photo browser [Sillberg et al., 2013] – or a combination of multiple back ends could enable the use of photo metadata and social profile information for the purpose of providing tag suggestions [Rantanen et al., 2017]. Even though the back ends can provide many different kinds of data and offer a wide variety of capabilities, from the architectural point of view these features should be – if not irrelevant – at least not the defining factor. The architecture should be capable of abstracting a feasible core communication layer for transferring various types of content. Still, the generality of the architecture can only be validated by testing the approach on as many different kinds of back ends as possible. The provided solution could work on other systems that require task delivery to back-end servers, but as the scope of this thesis lies in the content analysis domain, validation and testing is only performed with back ends especially designed for the purposes of content analysis.

As Figure 1.1 illustrates, in a very high-level use case, all the communication can be simplified to consist of tasks created or indirectly requested by the client, and to the results provided by the back end. In a simple, traditional client–server case consisting of either one or more clients and a single back-end server, there is no need for a separate front end server, but if multiple back ends are used some kind of control entity is required. The front end can either work as a load-balancer or perform more advanced functions such as relaying tasks to different back ends based on the capabilities of the back ends or the front end can function as a metadata index or cache for search results. The front end and back ends can be located on the same physical server, be divided over multiple servers, or consist of a more complicated cloud-based solution, but regardless of their internal organization these components can be thought to form a single service entity, as shown in Figure 1.1.

Figure 1.2 illustrates a high-level architecture with the common participants required for a functional system in the content analysis domain. As mentioned above, in this case clients can be anything from mobile phones to desktop computers. The operating systems and even client software implementations – whether native applications or web browsers – are irrelevant in this case. This “irrelevancy”, depending on the point of view, either allows the easy development of client software in combination with cross-platform and interoperability functionalities, or can be seen as one of the requirements for the system

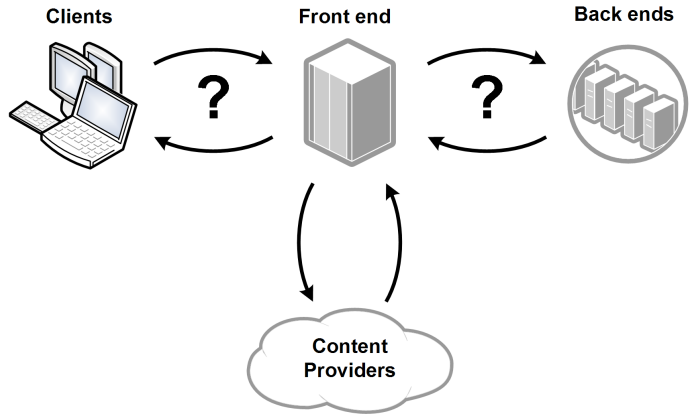

Figure 1.2: High level architecture diagram for content analysis environment with unknown interfaces described.

design. Generally, in the client–front-end (or client–server) communication, the methods of communication can be dictated by the front-end design, though they must be sensible enough to be usable in practice. In fact, the guidelines (best practices, protocols, data formats) for the client–front-end communication design are one of the more important issues in architectural design.

The back-end implementations (on the right, in Figure 1.2) partly dictate the requirements for the interaction between the front end and back ends, although regardless of the capabilities of the back ends a functional and syntactically appropriate, yet sufficiently generic communication protocol and data format should exist. If no common protocol between the front end and back end can be designed, the other option is to design a separate protocol for each back end, which can be a troublesome and time-consuming process both in the design and implementation phase, and in the future because of the increased maintenance complexity. In addition to the client–front-end communication, the front-end–back-end communication (best practices, protocols, data formats) is one of the cornerstones of the architecture.

Comparing Figure 1.2 with Figure 1.1, we can see that a new entity, the *Content Providers*, has been included. Content providers is a term, which is used to describe all third party services used in the design. For example, these can be image content services, which host the user's photos, or perhaps there is a social media website the user commonly uses to post his/her daily activities, or a third party service that could be utilized to authenticate the user. From a design point of view there is very little freedom in the design and implementation of the front-end interfaces that communicate with content providers – the providers must be accessed using the interfaces defined by the providers. Nevertheless, the providers play a crucial role in the overall design as the hosts of the user's content [Sillberg et al., 2013][Publication IV]. Due to the limited possibilities on how the communication with the *Content Providers* can be affected, the primary focus of this thesis is on the interaction of the front end with the clients and with the back ends when defining the protocols and communication methods. The interfaces and communication channels of interest are marked with question marks in Figure 1.2.

1.2 Research Questions

This thesis claims that it is possible to design a generic architecture and to define best-practice guidelines for the implementation of complex content analysis and metadata processing architecture. The guidelines are not strictly limited to content analysis and can be applied to any system design concerned with the problem of connecting incompatible back ends with external service providers in a client–server or cloud architecture. However, in the context of this thesis the validation of the architecture is performed in the content analysis domain utilizing the back-end implementations (PicSOM [Aalto University, Department of Computer Science, 2015], Kansei [Kiyoki and Chen, 2009; Kiyoki et al., 1994], MUVIS [MUVIS, 2015] and Social Media Summarizer [Forss et al., 2014]) developed and refined in the projects that the author of the thesis participated in. Based on these claims, and the scenario described in the previous section 1.1, the following research question – or problem – can be constructed:

Is it possible to define a generic architecture, which describes how clients, back ends, and content providers can be connected in a meaningful way?

Moreover, this doctoral thesis will present a generic architecture and show its feasibility through proof-of-concept implementations. This work will not only answer whether it is possible to implement a generic architecture, but will also specify how the architecture should be created and what components should be included in a practical design. Thus, the problem can be further refined into three research questions, which in this context can also be thought to be the primary requirements for the architecture. These questions are listed below:

1. ***What is the optimal method of communication for syntactically incompatible back ends?***
2. ***What are the crucial interfaces and data formats required for cross-platform communication?***
3. ***What kinds of aspects are required to guarantee system maintainability?***

In the scope of this thesis, and in relation to question 1 above, the solution will be considered optimal when it works as intended in the content analysis domain and the solution can be utilized to create a feasible and practical system. Content analysis engines may have been designed for various syntactically and contextually incompatible tasks, and it might be difficult to foresee how the data generated by the engines could be represented or used in an interoperable and generic way. For example, a back end designed for processing the user's daily action in a social media service can produce quite different data than another back end designed for audio analysis. To make the different use cases compatible, a common interface definition is required, which may even require changes to the original designs of the analysis engines.

In addition to the specific requirements imposed by the content analysis domain, the proposed architecture must comply with commonly accepted software quality requirements [ISO/IEC, 2011a] in order to be a valid software design paradigm. Characteristics such as *usability*, *security*, *maintainability*, *scalability*, *adaptability*, and *portability* are important considerations in the overall architectural design.

1.3 Thesis Structure

Throughout this thesis, two separate conventions are used for referencing source material. This has been done to help the reader more easily separate the two different "origins" of the source material. The publications included directly in this compilation thesis are referred to by the term *Publication* followed by a Roman numeral (e.g., [Publication I]) referring to a publication listed in chapter 3, whereas the material listed in the References section at the end of this thesis is referred to using a different style (e.g., [Rantanen and Sillberg, 2009]), and in general consists of material published by other authors or of supplementary material published by the author of this thesis.

Chapter 1 presents the content analysis ecosystem and its main participants and describes the starting points for the architectural design. The chapter also presents the research questions and the primary goals of the studies and concludes with section 1.3, which explains the structure of this thesis.

Chapter 2 explores the related technologies, existing solutions, and crucial points of interest in designing a generic architecture for the content analysis domain. The chapter also outlines certain limitations on the topics and technologies discussed, and thus narrows the scope of this thesis.

As this thesis is a compilation thesis, the included publications and their relation to the topic at hand have to be clearly presented. Chapter 3 explains the research method utilized to achieve validity for the claims of this thesis and presents the publications (scientific peer-reviewed papers) directly included in this thesis and also explores the other related material – published both by the author of this thesis and by other authors – that support the claims.

Chapter 4 presents the primary findings of the studies. It also expands the existing findings by providing a more in-depth look at the reasons for choosing the presented methods and proves the validity of the methods through quality attribute evaluation. The chapter also presents the API and data models formed based on the studies.

Finally, chapter 5 offers a look at possible future topics and directions for research, and concludes this thesis with a short summary of the main results.

2 Background Research

This chapter will present a look into existing research on the topic of architectural design and highlight the solutions and methods available for the realization of a practical content analysis architecture. The four important pillars (APIs, communication methods, data representations, and documentation) for implementing software architecture are described in the following subsections in the form of the research framework (presentation of the crucial components of a generic architecture).

2.1 Research Framework

There is no single, clear definition of what *software architecture* is. Architecture can be considered to be a set of structures, defining the software elements and the relations between them, and describing the properties of the elements and their relations [Bass et al., 2012; ISO/IEC, 2011b; Navon and Fernandez, 2011; Perry and Wolf, 1992]. Architecture can also be seen as a set of design choices [Bosch, 2004; Hofmeister et al., 2005]. The design of a good software architecture can yield substantial benefits in software management, reuse, construction, and analysis [Valipour et al., 2009], though the design also requires careful study, good tools, disciplined use, and proper documentation [Bosch and Molin, 1999; Perry and Wolf, 1992; Taylor et al., 2009].

There have been many studies on software architecture [Kruchten et al., 2006] in general, but in the context of this thesis the discussion on software architecture is limited to systems (or services) that operate in networked environments. Architecture is defined by a collection of well-defined APIs - contracts that specify the input and output parameters and methods required for communication as well as the structures and types utilized to represent the data.

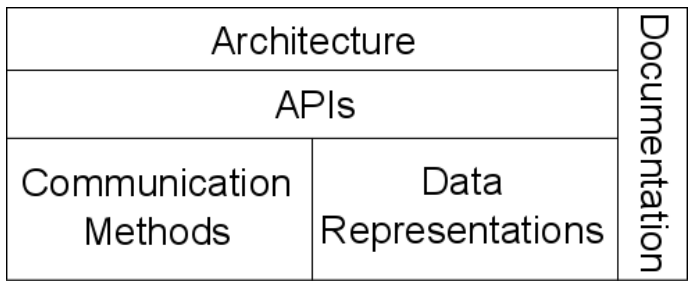

Figure 2.1: The architecture structure and its primary components.

Figure 2.1 illustrates the relations between *Architecture*, *APIs*, *Communication Methods* and *Data Representations*. On the top is the *Architecture* itself, which consists of any

number of APIs. The collection of APIs generally utilize multiple *Communication Methods* both for communication within the system in question and with interfaces located on external systems. For any communication to take place in a well-defined way, a collection of *Data Representations* must be specified. Finally, the documentation can be used to describe and specify the usage and design choices for the four other parts. In principle, the framework presented in Figure 2.1 could be expanded to include a more detailed description of a software architecture, and the model is not meant to be a conclusive representation. This thesis will not attempt to re-define the concept of software architecture, but will instead concentrate on the solutions and methods required to implement the practical interfaces required for the realization of a feasible content analysis environment. Thus, Figure 2.1 can also be regarded as a visualization of the scope of this thesis within the field of architectural design.

The existing commonly used solutions for the aforementioned components are briefly presented in the following subsections 2.2, 2.3, 2.4 and 2.5, which also describe certain limitations of the aforementioned topics in relation to the scope of this thesis. The figure also shows the role of documentation, which, if properly written and maintained, should describe the connections between the various components in a uniform format and style. Documentation is further discussed in subsection 2.6.

The five parts of Figure 2.1 can be considered the cornerstones of an architecture, and thus, each of these parts are further discussed in the context of the architectural solution presented in this work: the higher level architecture and communication methods are presented in section 4.1; the chosen API structure is explained in sections 4.3 and 4.4; data representation is discussed in section 4.5; and methods utilized to improve documentation can be found in subsection 4.1.5.

2.2 Architectures

Traditionally, architectures for networked systems have been described using commonly used architecture patterns (e.g., client–server, layered or multi-tier pattern) [Bass et al., 2012; Harrison and Avgeriou, 2007]. The content analysis domain is divided into clients (the users of the service) and servers (the providers of the service), to put it simply. Behind the scenes, the service often consists of a more complicated structure, perhaps of one located in a cloud and consisting of several back-end systems performing the required content analysis, storage, and retrieval operations. How well the designed architecture fits the task at hand can be measured - in addition to performance and use tests - by evaluating the architecture against a pre-defined set of quality attributes [Bass et al., 2012; Harrison and Avgeriou, 2007]. The quality attributes can vary between use cases, but certain common attributes always exist for networked systems, such as interoperability, availability, reliability, and security. Thus, the quality attributes - and their relation to the chosen approach - are discussed in detail in section 4.2.

There are four primary directions for implementing networked (in this case, web) service architectures: RPC [Birrell and Nelson, 1984; IETF, 1976; JSON-RPC Working Group, 2015; W3C, 2007a], REST [Belwasmi et al., 2011; Fielding, 2000; Fielding and Taylor, 2002; Pautasso et al., 2008; Richardson and Ruby, 2007], hybrid, and what could be referred to as a “proper” web service [W3C, 2004a]. In the literature, the term RPC is often analogous with web services, even though the term [Birrell and Nelson, 1984] predates the idea of modern web service, and the idea of remote procedure calls is much

older [IETF, 1976]. In the context of this thesis, the term RPC is in its more generic meaning - methods for delivering workloads or executing operations on remote peers.

As defined by W3C, a web service usually contains, in addition to the functional layer, a descriptive API, which can be used for service discovery [W3C, 2004a], which may or may not be present for RPC, REST or hybrid approaches. In fact, the technologies for implementing a web service are not limited and implementation could be achieved by utilizing any one of the three other architectural styles, although in principle the lack of certain supportive functionalities (namely, the descriptive interfaces) dictate that not all "web services" are (strictly speaking) *web services*. There has been research on descriptive interfaces for REST, although no conclusive, standard, or even commonly used approach exists [Ludwig et al., 2009; Pautasso, 2009; Verborgh et al., 2014]. The web service approach can be and often is used to implement services based on Service-Oriented Architecture (SOA), and the approach does have certain advantages, such as, improvements in interoperability, location transparency, composability, modularity, self-containment, scalability, and security [O'Brien et al., 2007; Valipour et al., 2009].

The often mentioned disadvantages of web services are the steep learning curve caused by the apparent complexity of implementing a proper web service, and the increased latencies caused by the overhead in message headers (e.g., SOAP headers) or by the increased processing time of the commonly used representational formats (e.g., XML or JSON). The message processing overhead and performance depend on the use case, and may or may not be increased by selecting a different architectural style, but the service design and implementation complexity are affected by the architectural choices. In general, REST is considered easier to learn and use, and that has perhaps been the primary reason for its increased adoption. [Aihkisalo and Paaso, 2012; Feng et al., 2009; Guinard et al., 2012; Mulligan and Gračanin, 2009; O'Brien et al., 2007]

Looking at the illustration of the content analysis use case in Figure 1.2 (section 1.1), we can see two separate scenarios: *Clients - Front End* communication and *Front end - Back ends* communication. The third scenario would be the communication between *Front end* and *Content Providers*, though this is more clearly (and strictly) defined because the communication is specified solely by the providers. For both scenarios, RPC implementation is possible, as is a web service based approach. However, both of these solutions add a layer of complexity to the implementation. For the second scenario, the complexity - if not entirely desired - is manageable, but for the first scenario, it is problematic. Especially if the clients have performance limitations (e.g., certain mobile devices), overhead in the communication should be minimized. More importantly, if the client side API is designed to be a public one, the complexity can have a detrimental effect on the developers' interest in using the system.

A better choice for the first scenario is to use the REST architectural style. Unfortunately, there are certain limitations in the second scenario, which prevent the use of REST. The REST style has the requirements of statelessness, representation of resources by Uniform Resource Identifier (URI) and the use of Create, Read, Update and Delete (CRUD) operations [1] - in addition to a number of other requirements described by other studies [Fielding, 2008; Li and Chou, 2011]. Of the three, statelessness can be difficult to achieve because there may be a need to track the status of operations running on the back

[1] Fielding's original work [Fielding, 2000] does not mention CRUD by name, the defined architectural style simply requires that all methods are uniformly defined for all resources. In the context of this thesis it is assumed that all operations use the methods of Hypertext Transfer Protocol (HTTP) in a well-defined CRUD way.

ends, and the back ends in turn must keep track of callback URIs used for returning the analysis responses. In other words, the operations are by nature stateful and more RPC like. Similar observations about the superior applicability of RPC (or web services) for distributed environments have been made by other researchers [Castillo et al., 2012].

Another option for the second scenario is to use a REST/RPC hybrid [Richardson and Ruby, 2007]. In essence, by choosing the hybrid approach we would keep parts of the REST style (e.g., CRUD and resource URIs) to preserve simplicity, but ignore the restraints that do not fit the current use case (e.g., statelessness). Creating a hybrid can provide advantages, but in many ways the process is equivalent to the creation of a custom design, increasing the need for proper documentation and examples.

2.3 APIs

Any well-defined service provided by one component, module, or application for other software elements can be considered to be an API [de Souza et al., 2004], and is an important tool in realizing modularity and information hiding in software design [Parnas, 1972]. The importance of API design has been highlighted in several studies and literature on best practices or guidelines for API design exists [Daughtry et al., 2009; de Souza et al., 2004; Espinha et al., 2014; Henning, 2009; Li et al., 2013]. In general, a good - or appropriate - API should be minimal (only functions that are required for a certain task should be exposed), stable (changes cause problems for developers), and properly documented.

In the scope of this thesis, APIs are limited to interfaces (or services) that are provided for communication over a networked medium. Furthermore, the communication between *Clients* and *Front End*, as illustrated in Figure 1.2 (section 1.1) is assumed to take place using a commonly used protocol (e.g., HTTP), with the client devices working either on dedicated client software or by utilizing a web browser. The clients are an integral part of the environment, and existing literature on web API development, its limitations and advantages, is available [Anttonen et al., 2011; Li et al., 2013; Paulson, 2005; Smith, 2006]. The design of the client and user interface (including the separation of User Interface (UI) from the APIs) is beyond the scope of this thesis, but is briefly discussed in relation to the architecture presented in this thesis in publications [Sillberg et al., 2013], [Rantanen et al., 2017], Publication IV and Publication VII.

2.4 Communication Methods

Based on the existing studies, there do not seem to be any commonly used generic and free protocols for task scheduling to external systems [Publication IV]. The design of a lower, network, or transport, level protocol is a non-trivial task, and there is very seldom any need to entirely re-invent the wheel, and for this reason low level protocols are not discussed in the scope of this thesis. In general, a standard stack of Transmission Control Protocol / Internet Protocol (TCP/IP) is assumed to exist. Additionally, HTTP is increasing in popularity in networked systems even outside its original purpose (transfer of hypermedia documents) and in the context of this thesis it is considered a viable choice for all communication.

In practice, the unknown duration of a single content analysis task requires the front end - back end communication to be asynchronous [Publication II]. There are several patterns (e.g., polling, message queue, result callback) for achieving asynchronous communications

[Brambilla et al., 2004; Zdun et al., 2004], with the usage of callback URIs for result submission being the most natural choice in the content analysis case, as described in Publication IV.

The client—front-end communication can utilize both synchronous and asynchronous communication, but synchronous communication can be easier to implement. As presented in Publication IV and Publication VII, the architecture will primarily index, process, and deliver metadata, with the actual content located elsewhere. Commonly, the metadata will provide Uniform Resource Locator (URL)s for the actual content, which is downloaded from a web server or from a larger Content Delivery Network (CDN) - i.e., any content delivery system that can be accessed utilizing the resource URIs [Dilley et al., 2002; Guo et al., 2005; Pallis and Vakali, 2006] is applicable with the architecture presented in this thesis. The actual content delivery is in itself a large and much researched field, and is not the primary focus of the research in this thesis.

2.5 Data Representation

Standards exist for multimedia content representation (e.g., ISO 15938 [ISO/IEC, 2002]), which might work without the need to define a new format from scratch. Depending on the use case, the existing solutions might be enough, and regardless of the problem domain, there are many standardized formats for commonly used data types (e.g., UTF-8 [ISO/IEC, 2014; Unicode, Inc., 2016] for character encoding or ISO 8601 [ISO/IEC, 2004] for date and time).

Creating an entirely custom (or proprietary), even binary based, format can deliver a performance advantage when compared with commonly used formats. The disadvantages are the design process - which depending on the complexity of the data might not be trivial - and the increased work on parser implementation and maintenance compared with the use of existing libraries. Using an existing library does not necessarily guarantee better security, but with publicly available solutions the common security problems are often resolved.

The two formats commonly used for data representation are JSON [Ecma International, 2013; IETF, 2014] and XML [W3C, 2008]. Parsers for both formats are available (practically) on all mobile and desktop devices and programming languages [json.org, 2016; O'Reilly Media, Inc., 2016], making either one a viable choice. JSON is generally considered to perform slightly better, although the difference between the two formats is not significant on modern devices with the major performance factor being the parser implementation used for data serialization [Aihkisalo and Paaso, 2012; Riva and Laitkorpi, 2009][Publication III].

The proof-of-concept implementations presented later in section 4.6 generally utilize XML although this is primarily for historical reasons (reuse of existing software components) and the developer's preference. The excellent availability of libraries and frameworks also makes it generally redundant to offer both XML and JSON, but it also makes the choice between the two a difficult one. This thesis does not recommend either one over the other, or even completely rule out the development of a proprietary format, as depending on the use case each approach may be valid.

2.6 Documentation

Documentation is important for any software component and the architecture as a whole is no exception. The design rationale of the architecture should be well documented in addition to the defined quality attributes [Bass et al., 2012; Clements et al., 2010; Tang et al., 2006]. Research also suggests that among the most common documentation artifacts for APIs are interaction examples (e.g., client–server communication examples) [Danielsen and Jeffrey, 2013; Maleshkova et al., 2010], in addition to the source code itself [de Souza et al., 2005]. However, keeping examples up-to-date can be one of the most tedious parts of the documentation, especially if continuous development or similar methods that require constant documentation updates are utilized. In general, automated documentation is preferred, though for complex systems, such as the content analysis environment, achieving a fully automated solution might be problematic [Publication VI].

Based on the aforementioned research it can be concluded that documentation has two distinct, but in no way mutually exclusive tasks. Firstly, the documentation can improve the overall maintainability of the designed system by explaining the reasons for particular design choices. Comprehending the limitations and possibilities of the design can also improve the extendability and adaptability when the system evolves through future (feature) additions and software bug fixes. Secondly, documentation can help the users of the APIs - whether they are developers, part of the design team, or external parties - in understanding how the APIs work and should be used, improving the usability of the system.

The field of architecture evaluation has been studied rigorously. There exists comprehensive literature both for evaluating the architecture requirements and the evaluation process [Babar and Gorton, 2004; Bachmann et al., 2000; Dobrica and Niemelä, 2002; ISO/IEC, 2011b; Patidar and Suman, 2015; Roy and Graham, 2008] and for evaluation optimization [Aleti et al., 2013]. Additionally, Nord et al. [2009] have provided a list of sample question sets for reviewing whether the architecture conforms to several existing evaluation methods. In the context of this thesis, the evaluation methods are not extensively studied. This is primarily due to the fact that there are multiple projects with various internal evaluation methods, which do not necessarily directly relate to existing evaluation methods, making it challenging to carry out a comprehensive analysis of which method would be optimal. Instead, this work shows the validity of the generic architecture through proof-of-concept implementations as opposed to the existing evaluation methods generally utilized before and during the architecture design process.

Similarly, methods for documenting architecture have been studied by several authors. Authors such as Clements et al. [2010] (architecture views), Gerdes et al. [2016] (erosion of documentation), de Graaf et al. [2012] (ontologies), Medvidovic and Taylor [2000] (Architecture Description Language (ADL)), Emery and Hilliard [2009] (frameworks), Hadar et al. [2013] (documentation strategies for agile development) and Rost et al. [2013] (developer's perspective) have extensively studied how to document an entire architecture. In general, based on the related literature, the main problems of architecture documentation are largely the same as for the documentation of individual software components or smaller applications: documentation maintenance and relevancy are often lacking. A few authors have also proposed tools for the creation and management of architecture documentation [Bachmann and Merson, 2005; Emery and Hilliard, 2009; Jansen et al., 2009; van Heesch et al., 2012] as well as comparisons of commonly used documentation methods and strategies [Alexeeva et al., 2016; Kruchten, 2009; May, 2005].

This thesis will not attempt to provide a comprehensive comparison or a recommendation of which method to use. In principle, any method is applicable, but in practice, the complexity of the presented architecture makes it very challenging to test and analyze every approach. In any case, as the documentation is unquestionably an important part of the process of designing and implementing any software architecture, section 4.1.5 will describe the methods utilized in documenting the generic architecture (iterations) presented in this thesis.

3 Research Contribution

This chapter describes the methodology used to validate the claims presented in this thesis. The publications included in this compilation thesis are briefly introduced and their relations to the research topic and to the research method are explained. In sections 3.3 through 3.9, the publications are presented in chronological order with the oldest publication first. The descriptions contain short summaries of the main content of the publications, the author's contribution per publication can be found at the beginning of this thesis (in Author Contribution). The relation of the publications to the research questions is further explored in the sections below with an illustration of the chronological order of the publications and the studies in general.

3.1 Timeline of the Studies

The studies originated in the Seamless Services and Mobile Connectivity in Distributed Disaster Knowledge Management (*SSMC/DDKM*) project. The alert message delivery system designed in the project provided the first specification – and was also the first evolutionary version – for the generic architecture, and the design of the system was also the topic of the Master's thesis of the author. The architecture was extended and improved throughout the projects that followed the initial *SSMC/DDKM* project. Thus, the research included in this doctoral thesis is greatly influenced by the work performed over several years in the projects in which the author of this thesis has participated. The projects provided not only the tools and the resources that helped to realize the practical work, but also created a medium for receiving feedback from the industry and educational partners participating in the projects – a medium, which in part guided the directions of the research presented in this thesis.

The author's projects and their duration are illustrated on the timeline in Figure 3.1. In the figure the publications included in this thesis can also be seen in relation to the projects, and color coding is used to separate the projects from one another. The figure also shows three other important milestones in the design process: the starting date of the architectural development in 2008 when the author started as a researcher at Tampere University of Technology; the author's acceptance to the doctoral program in 2011; and the date of publication for this doctoral thesis in 2017. Interestingly, two of the publications (Publication I and Publication II) were published before the author was accepted to the doctoral program (i.e., started his doctoral studies), and thus, they were also published before the research questions – or the research problem – presented in this thesis were formulated. Nevertheless, their inclusion in this thesis is well justified because of their inherit relationship to the topic of this thesis – the aspect of architectural design and the interfacing of complex systems. The work – both the background studies on architectures as well as the implementation of the proof-of-concept prototypes – created

important knowledge on how to design and implement a practical system. In other words, the work is an integral part of the evolution of the architectural design presented here, and the evolution is further explored in section 4.1.

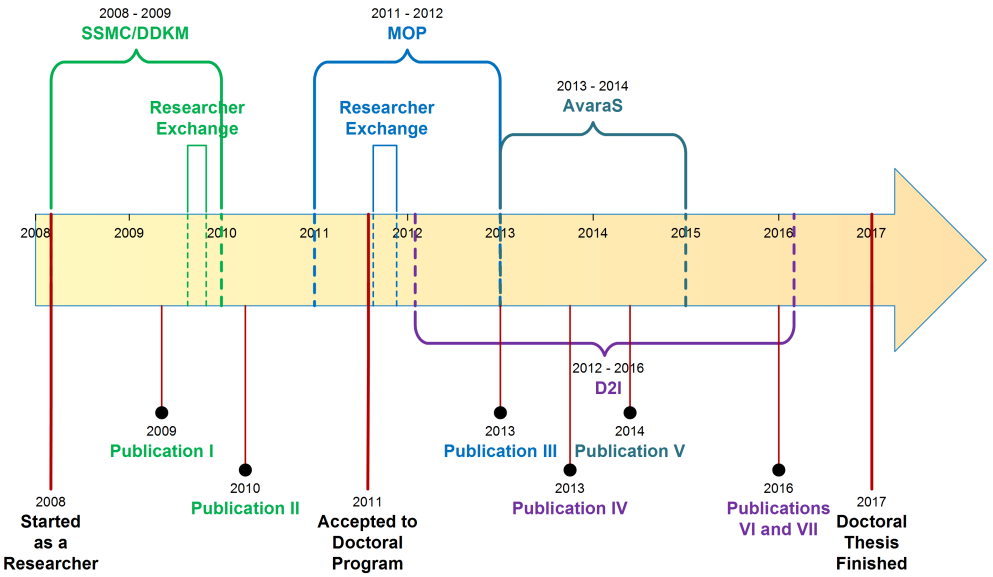

Figure 3.1: Timeline of the research with important milestones and publications.

In 2009 and 2011, the author of this thesis participated in a researcher exchange with Keio University Shonan-Fujisawa Campus (SFC), and worked for a total of six months in Japan. During these visits the author had the opportunity to study the Kansei-based [Nagamachi, 2010] semantic search and information system [Kiyoki and Chen, 2009; Kiyoki et al., 1994]. The first visit also included an internship awarded by the NICT of Japan, which made it possible to obtain information on the network and computer architecture utilized in the design and implementation of the Knowledge Grid used as the deployment platform for the Kansei-based system. Also, the results of the researcher exchange period were reported in scientific publications [Publication II][Soini et al., 2011][Publication III]. The work done in co-operation with NICT and Keio University included both the design of the first integration specifications for the architecture and also provided the first test back end for analysis benchmarks.

The Mukautuvat OhjelmistoPalvelut (*MOP*, "Adaptive Software Services", in English) project, the Avoimista tietoVARAnnoista liiketoimintaa Satakuntaan (*AVARAS*, "From Open Data to New Business in the Region of Satakunta", in English) project, and the Data to Intelligence (*D2I*)[Digile, 2015a] program continued the iteration process for the generic architecture. The most important influence came from the *D2I* program, which helped to form the final proof-of-concept implementation (see subsection 4.6.3), and provided significant contributions to the development of the analysis back ends utilized for testing and benchmarking purposes. The MUVIS [MUVIS, 2015] system designed by the Tampere University of Technology Department of Signal Processing (SGN), the PicSOM [Aalto University, Department of Computer Science, 2015] system designed in Aalto University, and the social media summarizer [Forss et al., 2014] from Arcada University of Applied Sciences have been used as the test-use content analysis back ends.

Also visible in Figure 3.1 is the short gap between the first two projects (*SSMC/DDKM* and *MOP* caused by the author's participation in another research project unrelated to the topic of this thesis.

3.2 Research Method

The research method used in this study is "Design Science" [Fuller, 1957; March and Smith, 1995; Simon, 1996]. Especially in the context of information systems, the purpose of design science is to create innovations that define ideas, practices, technical capabilities and products, which help to analyze, design, implement, manage, and use the systems more effectively and efficiently [Denning, 1997; Hevner et al., 2004; Tsichritzis, 1997]. The result of design science research is an artifact, which will provide a solution for the formulated problem, and its research should provide new data on the research topic itself [Hevner et al., 2004]. The artifact is constructed for a specific function, and the construction shows that the design problems have been resolved [March and Smith, 1995]. The resolution of the problems is in essence the process of application, testing, modification, and extension of existing kernel theories through experience, creativity, and intuition [Hevner et al., 2004; Markus et al., 2002; Walls et al., 1992]. In this thesis, the artifact of the design science is the proposed generic architecture, and this thesis will describe the design process – the evolution of the architectural iterations (section 4.1) – and the existing technologies utilized in its construction (chapter 2).

In addition to the design process, an important factor in the creation of the artifact is the validation process, which is typically based on case-specific performance metrics [Hevner et al., 2004]. To validate the generic architectural design, the crucial metrics – or quality attributes (section 4.2) – of the system need to be considered and the overall design should be validated to conform to the requirements. The advantage of executing multiple projects while designing the architecture is in the feedback achieved from a wide audience – from internationally peer-reviewed conferences and publications as well as from people involved in the research projects. The disadvantage is that in every project the validation, risk assessment, code reviews, and background research procedures have been individually decided with the project participants in accordance with the official project guidelines. The procedures have, in general, closely resembled those of the Architecture Tradeoff Analysis Method (ATAM) [Kazman et al., 2000], although the variations in the project procedures have made it challenging to make a precise analysis of the architecture's usability based on any single evaluation method. Thus, this thesis will validate the architecture through proof-of-concept implementations as opposed to utilizing an existing validation method.

The relevancy of the research must be clearly presented with respect to the target community [Hevner et al., 2004]. In the scope of this thesis, the community consists both of various industrial and educational partners participating in the research projects as well as potential developers of systems that utilize complex content analysis systems. The relevancy of the presented research has been shown by studying the existing literature, which has an apparent lack of studies on designing architecture for content analysis environment (as described in section 1.1. Additionally, as the research has been carried out over multiple research projects, discussions with the industrial and educational project partners have indicated the importance of the content analysis research, and of the design of a practical generic solution.

For the purpose of systematically validating the research approach of this thesis, the seven

guidelines created by Hevner et al. [2004] for assisting researchers, reviewers, editors, and readers to understand the requirements for effective design-science research have been followed throughout the research process. The guidelines and their meanings, in relation to the context of this thesis, are summarized below:

1. *Design as an Artifact.* The primary outputs of design science in information systems research are purposeful artifacts created to address important organizational problems. The artifacts must be described effectively, enabling the implementation and application of the artifacts in appropriate domains.

2. *Problem Relevance.* The objective of research in information systems is to acquire knowledge and understanding that enable the development and implementation of technology-based solutions to previously unsolved and important business problems.

3. *Design Evaluation.* Design is an iterative and incremental activity, and the design artifact is complete and effective when the pre-defined requirements have been fulfilled. The utility, quality, and efficacy of a design artifact must be demonstrated through the design process.

4. *Research Contributions.* Often, the contribution of the design-science research is the artifact itself (for example, a prototype system), which may create new knowledge or apply existing knowledge in new and innovative ways. New value can also be created through innovative use of evaluation methods or by extending existing methodology.

5. *Research Rigor.* An important aspect is the effective use of existing theoretical foundations and research methodologies, and appropriate metrics should be applied to validate the created artifact.

6. *Design as a Search Process.* Design is essentially an iterative search process to discover an effective solution to a problem by utilizing the available means.

7. *Communication of Research.* The artifact, how it can be applied, and how it is to be constructed (implemented), must be presented to the appropriate (technology-oriented and management-oriented) audiences.

To better illustrate how the guidelines have been utilized in the research presented in this thesis, Table 3.1 below shows the guidelines, and their relationship to the three research questions presented earlier in section 1.2. For reference, and for the reader's convenience, the research questions are also repeated below:

1. *What is the optimal method of communication for syntactically incompatible back ends?*

2. *What are the crucial interfaces and data formats required for cross-platform communication?*

3. *What kinds of aspects are required to guarantee system maintainability?*

In Table 3.1, in addition to the publications included in this thesis, several other references can be found. Some of these refer to work ([Malik and Nieminen, 2014] presented by other authors [Malik and Nieminen, 2014], and for others ([Sillberg et al., 2013], [TUT Pori,

2016c][1] and [TUT Pori, 2015b]), the author of this thesis was one of the co-creators, but the participation was not significant enough to justify the inclusion of the work in this thesis. Nevertheless, all these publications – directly or indirectly – prove the validity of the generic architecture explained in this thesis. Most of the presented publications use case studies, user trials, or prototype applications that were used to test a part or an aspect of the generic architecture, or describe a proof-of-concept implementation based on the architecture. Thus, being performed by pilot users or other researchers, these test cases can perhaps prove the validity of the approach even better than the tests executed by the author of this thesis – or at the very least complement the author's own observations.

Table 3.1: Design science research guidelines [Hevner et al., 2004] in relation to the research questions.

	Question 1	**Question 2**	**Question 3**
Guideline 1: Design as Artifact	Present a publicly available generic architecture that assists in fulfilling the quality attributes (as described in subsection 4.2) required for a practical system operating in the content analysis domain.		
Guideline 2: Problem Relevance	Based on the review of existing literature, there is no open and publicly available existing generic architecture for integrating clients, back ends and content providers in the content analysis domain.		
Guideline 3: Design Evaluation	Proof-of-concept implementations of the architecture [Publication III][Sillberg et al., 2013][Publication IV][Publication V] and Publication VII on VisualLabel platform, and user trials [Malik and Nieminen, 2014]. Release of the implementation as an open source [TUT Pori, 2015b] and deployment of the service on a cloud platform [Digile, 2015b].		The generic architecture should be capable of testing and deploying itself, as described in the VisualLabel Deployment Guide [TUT Pori, 2016c], and generating up-to-date documentation through automation, as discussed in the Publication VI.
Guideline 4: Research Contributions	The author's publications and this thesis provide guidelines for the design of the generic architecture.		
Guideline 5: Research Rigor	The design of the architecture should adhere to commonly used software design paradigms and quality requirements [ISO/IEC, 2011a].		
Guideline 6: Design as a Search Process	The architectures described in Publication I, Publication II, Publication III and Publication IV can be thought as the evolutionary path to the ultimate goal.		
Guideline 7: Communication of Research	Publication II, Publication III, Publication IV, Publication VII, and this thesis.	Author's publication [Sillberg et al., 2013], Publication IV and Publication VII, and this thesis.	Publication VI, the VisualLabel Deployment Guide [TUT Pori, 2016c], and this thesis.

The most important part of any design science research is the design process [Hevner, 2007]. Thus, this thesis will describe the evolution of the generic architecture (section 4.1) as well as presenting the proof-of-concept implementations (section 4.6) to better illustrate the path taken to achieve the final version of the architecture. Arguably, the presented architecture could be further improved through additional iterations and further development, but in general, a software development process can be considered to be "finished" when the software solution fulfills its pre-defined requirements (see section 4.2).

[1]The VisualLabel framework was also deployed on Digile's (now Dimecc) FORGE cloud service[Digile, 2015b]. Unfortunately, the discontinuation of the cloud platform in 2015 [Dimecc, 2016] has made the presentation videos for the VisualLabel implementation on the FORGE platform unavailable.

Furthermore, the study has followed the six steps of the mental model proposed by Peffers et al. [2007], which provides a nominal process for the conduct of design science research. The model does not require the research to follow the steps in order. In the case of this research, the foundations of the generic architecture originate from earlier studies, which had different goals (the creation of an emergency messaging system), and later, the research topic focused on the content analysis domain. The case is referred to by Peffers et al. [2007] as the *design and development* approach with the logical starting point being the third step of the model. In other words, the overall research can be considered to have started from step three, although the process of realizing this thesis started from step one, after the goals and the research domain (content analysis) had been chosen. Additionally, the last step (Communication) was performed continuously during the progression of the research in the form of publications of the research work. The original steps of the mental model, and their applicability to the scope of this thesis are listed below:

1. Problem identification and motivation.
2. Definition of the objectives for a solution (study of existing frameworks and technologies to gain an understanding for the requirements for the architecture).
3. Design and Development (specification of the capabilities and interfaces for the architecture).
4. Demonstration (execution of use case test scenarios and user studies to find out the usability of the designed architecture).
5. Evaluation (analysis of the collected data).
6. Communication (submission of scientific papers to international conferences to receive feedback on the designed architecture and finally, compilation of the doctoral thesis).

These steps have been adapted to the context of this thesis, with the adaptations presented in parenthesis. The steps can also be identified within the structure of this thesis: the first stage can be found in section 1; the second stage is in section 2; the third stage is split between three subsections, 4.3, 4.4, and 4.5; the fourth stage is in sections 4.1 and 4.6; and section 4 describes the fifth stage. The sixth stage consists of this entire thesis and of the publications included in it, in combination with other published material as presented in Table 3.1. A synopsis of each publication included in this thesis can be found in this section (subsections 3.3, 3.4, 3.5, 3.6, 3.7, 3.8 and 3.9).

3.3 Publication I

Sillberg, P., **Rantanen P.**, Saari, M., Leppäniemi, J., Soini, J. and Jaakkola, H., "Towards an IP-Based Alert Message Delivery System", in Proceedings of the 6th International ISCRAM Conference, ISBN 978-91-633-4715-3, Gothenburg, Sweden, May 10-13, 2009.

Publication I presents a system architecture designed for the delivery of emergency messages to mobile terminals. The system can be said to represent the first iteration of the final generic architecture presented in this thesis. Several key architectural points of the system are utilized in the final architectural solution. Two different methods of communication are used, each designed for a specific goal – or, as in this case, target

user group. Unauthorized users access the messages provided by the system in a RESTful way by retrieving Atom-based feeds. Authorized users can take advantage of the RPC style SOAP-based communication method designed for bi-directional communication. Additionally, the system implements a method for utilizing external service providers and data sources. The overall system design is further explored in the author's Master's thesis [Rantanen and Sillberg, 2009].

3.4 Publication II

Rantanen P., Sillberg, P., Jaakkola, H. and Nakanishi, T., "An Asynchronous Message-based Knowledge Communication in a Ubiquitous Environment", Database Systems for Advanced Applications, Springer, ISBN 978-3-642-14588-9, DOI 10.1007/978-3-642-14589-6_44, pp. 434-444, 2010.

Publication II presents an extension for the system presented in Publication I (see section 3.3) designed for incorporating the use of external systems, in this case, the Knowledge Grid developed by NICT. The extension is implemented as a mediator service (the Grid Access Gateway), which works as a relay for translating the communication and data formats used by two separate systems (the Message Server and the Knowledge Grid). In the context of this thesis, the Message Server can be considered to represent the front-end service, the Knowledge Grid is an example of a back-end service, and the Grid Access Gateway is an implementation of a mediator API. Additionally, the Knowledge Grid shows many of the same features as the analysis back ends discussed in this thesis – namely, the "unknown" duration of the requested operations, asynchronous communications, generation or enrichment of existing metadata, and relatively loose coupling of the participating services.

3.5 Publication III

Sillberg, P., Kurabayashi, S., **Rantanen P.** and Yoshida, N., "A model of evaluation: computational performance and usability benchmarks on video stream context analysis", Information Modelling and Knowledge Bases XXIV, IOS Press (Frontiers in Artificial Intelligence and Applications; Volume 251), ISBN 978-1-61499-176-2, DOI 10.3233/978-1-61499-177-9-188, pp. 188-200, 2013.

Publication III presents an environment and proof-of-concept system for video content analysis, and describes the performance benchmarks used to validate the system. The purpose of the system is to extract metadata (keywords) that describes the "feeling" of the videos by utilizing an external video analysis back end (MediaMatrix) that is based on the Mathematical Model of Meaning (MMM) and color histograms. The paper illustrates a three-layer architecture – consisting of clients, front end (called "Web Service" in the publication), and back end – similar to the generic architecture presented in this thesis. The paper shows methods for describing back end analysis tasks and task responses, as well as client REST APIs by utilizing structured data formats (JSON and XML).

3.6 Publication IV

Rantanen P., Sillberg, P. and Soini, J., "Content Analysis System for Images", in Proceedings of the 16th International Multiconference Information Society, IS 2013,

Volume A, 7-11, Ljubljana, Slovenia. Josef Stefan Institute, pp. 241-244, October 7-11, 2013.

Publication IV presents a concept architecture developed for managing digital data using content-based analysis targeted to photographic content as an example use case. The paper illustrates how a front-end service can be utilized to provide media content for end-user (client) devices, and how users' media content can be delivered to back ends for content analysis and automatic metadata generation by utilizing a task-based approach for the delivery of analysis, search, and feedback workloads. The task-based solution described in the paper is a form of RPC communication and it is used in combination with a REST API targeted to client devices. A proof-of-concept implementation based on the task-based approach is presented in another publication [Sillberg et al., 2013], and both the task-based approach and the proof-of-concept implementation are designed based on the generic architecture presented in this thesis.

3.7 Publication V

Rantanen P. and Sillberg, P., "Event Calendar for Internet Data Sources", in Proceedings of the 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, Croatian Society for Information and Communication Technology, DOI: 10.1109/MIPRO.2014.6859721, pp. 1035-1040, May 26-30, 2014.

Publication V presents an event calendar service which can be used to combine data from multiple sources and represent the enriched content in a web client by means of a calendar view or a map view. The core platform used to implement the service is based on the same generic architecture model as the one presented in this thesis and the service itself can be seen as a proof-of-concept implementation. The primary focus of the paper is in the utilization of NoSQL databases in the context of the event calendar service, but it also discusses topics important to the topic of this thesis, namely, structured data formats (JSON and XML) and the realization of the calendar service concept by taking advantage of the features of the front-end service.

3.8 Publication VI

Rantanen P., "REST API Example Generation Using Javadoc", Computer Science and Information Systems, ComSIS Consortium, ISSN 1820-0214, 2017. *Accepted for publication.*

Publication VI describes how source code based service declarations and tools commonly used with the Java programming language can be used to automatically generate documentation and use case examples, either by taking advantage of existing interface declarations or, as in the case of the example presented in the paper, by implementing a simple Example/Reference API paradigm. The focus of the paper is on how to generate and update documentation, a goal that is important for successful software management and especially maintenance. The API model presented in the paper is part of the larger API model presented in this thesis (see 4.3).

3.9 Publication VII

Ahmad, I., **Rantanen P.**, Sillberg, P., Laaksonen, J., Liu, S., Forss, T., Malik, A., Nieminen, M., Shetty, R., Ishikawa, S., Kallio, J., Saarinen, J. P., Gabbouj, M. and Soini, J., "VisualLabel: An Integrated Multimedia Content Management and Access Framework for Personal Users", in Proceedings of the 27th International Conference on Information Modelling and Knowledge Bases (EJC 2017), Krabi, Thailand, June 5-9, 2017. *Submitted.*

Publication VII presents an integrated framework – called VisualLabel – for managing multimedia content by utilizing a front-end service in combination with multiple analysis back ends. Each individual back end is targeted to a specific task, such as the generation of keywords, summarization of a social media account, or content-based image retrieval. The article explains the common use cases for content analysis needs, the overall architecture, and a proof-of-concept implementation for the analysis, search, and representation of multimedia content, and it also provides background research for the studies related to CBIR systems. The article also presents a simplified version of the API model that is further explored in this thesis (see 4.3), and the core VisualLabel framework presented in the publication is based on the principles presented in this thesis. The paper also illustrates the common use cases required for realizing a feasible content analysis environment.

4 The Generic Architecture

This chapter presents the main results of the studies performed for this thesis, and discusses the conclusions made based on the observations. The chapter is divided into six sections, each describing an important aspect of the architecture. Section 4.1 describes the evolution of the (high-level) architecture as originally presented in the publications. A more in-depth look at the architecture is given in section 4.2, which defines and describes the crucial quality attributes related to the architecture and to the content analysis domain in general. The proof-of-concept implementations based on each of the architectural iterations are described - including an examination of the technologies chosen for each implementation – in section 4.6. Based on this observation, two models (API and data model) have been created that are presented in sections 4.3, 4.4, and 4.5.

4.1 Evolution of the Architecture

An important part of the Design Science approach is the search process (see 3.2), which in this case takes the form of iterative development of the generic architecture. The various implementations presented in the publications included in this thesis can be said to represent the evolutionary path of the final version of the architecture. To illustrate the path taken, this section will explain the architecture iterations, their primary components, modifications compared with previous iterations, and the reasons behind the implemented modifications. Many of the architecture diagrams included in this section can be found in the original publications included in this thesis, but in the publications the diagrams are designed and presented in different styles and icons. In this section, all diagrams are redrawn and illustrated using the unified annotation and icon style utilized in the most recent publication.

The reader should bear in mind that the iterations only represent the evolution of the architectural design – the implemented systems presented in the publications do not necessarily share the same goals, and were not used for identical purposes. Additionally, not all of the systems share the same codebase, and in fact, most of the systems were re-implemented (and re-designed) because of the lessons learned, improvement ideas formed, and difficulties faced in the process of constructing previous iterations. Regardless of the original goals of the iterations, they all represent integral steps in the overall evolution of the architecture and as such, provide a significant contribution to the design process as a whole. During the process of designing the iterations presented in this subsection, many more versions – or revisions – of the architecture were drawn, but here only the ones presented in the works published by the author of this thesis are presented.

4.1.1 The First Iteration

Figure 4.1 illustrates the initial version of the architecture as presented in Publication I. The primary design goal of the system was to deliver emergency messages to client devices and to enable traceability for message acknowledgements and communication failures. Additionally, the system provided the functionality for receiving additional emergency situation information from authenticated clients. The communication was to be performed by utilizing networks based on TCP/IP as opposed to more traditional means of communication (such as phone calls and text messages). These goals created some common requirements for system design. Firstly, there was a need for user authentication and the possibility that the users might use the service anonymously. Secondly, as the emergency messages originate from outside the delivery system (front end), perhaps from an emergency center or other applicable message provider, the design had to include the means for taking advantage of any number of external service (content) providers. All these requirements are still present in the final iteration of the design (section 4.1.4).

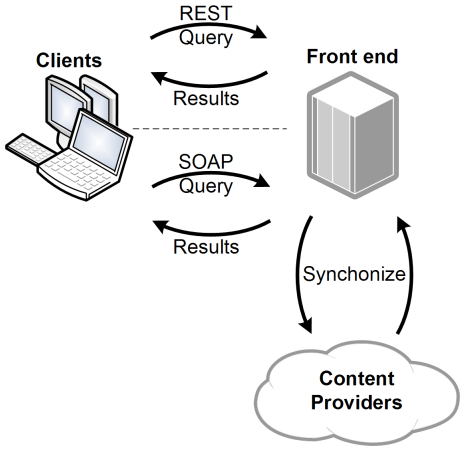

Figure 4.1: High level presentation of the architecture from Publication I.

In this iteration, the communication between the front end and the clients utilizes two separate communication methods: bi-directional communications encapsulated in SOAP messages, and one-way information retrieval. In the original publication, REST is not mentioned, but as the information retrieval uses standard HTTP GET operations in combination with a web feed, the implementation fulfills the requirements for RESTful communication. The separation of client-side communication into two ”channels” was created to allow easier access to public content (accessible by any web browser or Atom [IETF, 2005] capable Rich Site Summary (RSS) reader) whilst reserving the more secure SOAP-based implementation (accessible by the custom client software) for authenticated users.

The communication between the front end and content providers was designed to utilize content synchronization polling with provider-specific pre-defined polling intervals. If required, for example, for more urgent messages, the external services could also work as ”clients” for the system and post messages using the SOAP-based client interface. In principle, the content provider synchronization functionality has remained almost identical throughout the architecture evolution – the primary reason being the fact that utilizing the polling approach has been the simplest and fastest to implement. The communication

interfaces for the content providers are designed, implemented (and enforced) by the providers, and usually the architecture designers cannot affect the provider interfaces in any way. The only option is to create a reasonably generic synchronization mechanism with the possibility of extending the functionality by means of content provider specific adapters, which process and convert the information retrieved from the provider services to a format utilized by the front-end service.

4.1.2 The Second Iteration

The second iteration can be seen in Figure 4.2, and the corresponding implementation is explained in Publication II. The architecture reuses elements from the iteration presented in the previous subsection (4.1.1) with the same methods of communication present for clients and content providers, although in Publication II the content provider interfaces are not described. Nevertheless, as explained in the publication, the two-system iterations share the same codebase and features, including content provider synchronization.

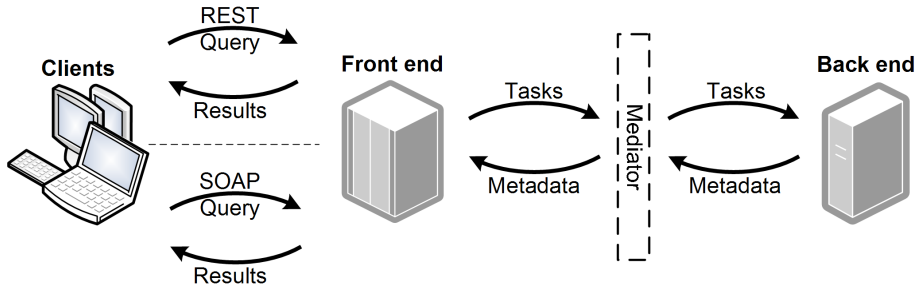

Figure 4.2: High level presentation of the architecture from Publication II.

The most important addition to the architecture offered by the second iteration is the concept of a back end. In a generic architecture, a back end is any service or system that provides a pre-defined set of external functionalities (or capabilities). In many ways back ends are a special case of content providers, as sometimes the content providers can also provide external services identical to those provided by back ends. One example would be photo analysis: when synchronizing the user's photo collection with the front-end service, it may be possible to retrieve metadata generated by the content providers, and this metadata can be similar in nature to the metadata generated by the back ends. The main difference between a content provider and a back end (in the context of the thesis) is the level of control in the design of the interfaces. While in the case of content providers, there is no real control on what features, formats, or technologies the remote interface can include, for the back ends, the front-end interface specification dictates the back-end implementation.

Publication II presents a use case where the NICT Knowledge Grid[Zettsu, 2012] works as a back-end service, processing tasks generated automatically based on the data given by the users. The presented case also illustrates the use of a mediator (or facade) service used for translating the front-end request and response formats (in this case, the tasks or workloads delivered to back ends) to make them compatible with the back-end formats, and vice versa. A mediator service does not necessarily have to be a separate service or located on dedicated hardware, but can also be part of the back-end service – for this reason the *Mediator* in Figure 4.2 is illustrated using a different style and icon than the other services (*Front end* and *Back end*. The mediator pattern is well-known in

object-oriented programming, and in this case it is simply extended to be applicable in the context of communication interfaces between various systems. The mediator pattern may be a valid approach when the back-end interfaces cannot be modified or the modifications are for some reason not preferable. Regardless of the case, to preserve the interoperability and uniformity of the front-end interfaces, the front end should not be modified on a back-end basis. The architecture does not limit the number of back ends, and thus, making back end specific modifications to the interface could in practice cause severe problems in interface maintenance and design.

In the original publication (Publication II), the presented architecture includes a slight design oversight. The messages the users post to the system are analyzed in a very back end specific way on the front-end service, even though content analysis is not the primary task of the front end, but that of the back ends. Of course, the message parsing in the case presented in the publication is quite simple, and some level of analysis does always take place on the front end (such as detecting content types and data sources), but in general, the analysis should be limited to discovering which back end should process the given content.

The communication between the front end and clients uses basic, simpler, and synchronized HTTP operations, but an asynchronous method was chosen for front end and back ends. The primary reason for the choice was an issue often encountered with content analysis: it is very difficult to estimate how long a specific operation will last, and in many cases, the operations can take a relatively long time. Asynchronous communications increase the overall complexity of the system (e.g., tracking of request-response pair identifiers, the use of callback URLs...), but in turn, they can free resources by keeping the connections active only when necessary.

4.1.3 The Third Iteration

From the architectural point of view, the third iteration offers only one alteration – the removal of the SOAP-based secure communication channel. The change was part of an attempt to simplify and unify the client interface, and provide all of the features through RESTful interfaces. Additionally, the third iteration marks the move from emergency messaging to the content analysis domain, which caused, or made possible, format changes on the client side and in part made the SOAP-based (more complex) interface redundant as all of the required features could be implemented utilizing a simpler, yet more customized XML based format. In principle, a standard or industry de facto format on the client- and back-end side would be preferable, but in practice, it might be difficult to find a suitable format, which was also the case with the content analysis domain.

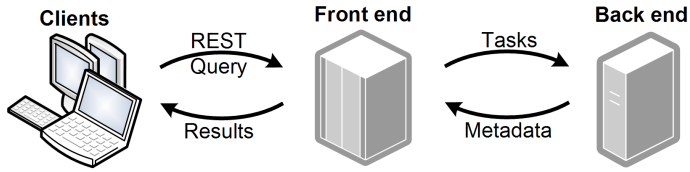

Figure 4.3: High level presentation of the architecture from Publication III.

On the back-end side, the third iteration attempted to mimic REST like simplicity, and although the interfaces turned out to be closer to traditional RPC – or one type of REST hybrid – the design proved to be a working setup. The main reason for this was the need

to maintain certain state information related to the progress of the tasks delivered to the back ends. Thus, the design took elements from the web service and SOAP world (such as callback URIs, identifiers, and status information), but utilized a more lightweight, but still XML based format. The back-end and client-side data types were unified for easier design, management, and implementation of parsers and data structures.

4.1.4 The Final Iteration

The final iteration is shown in Figure 4.4. All participants from the previous iterations are present – also, a mediator service could exist between the *Front end* and the *Back ends*, as illustrated in Figure 4.2, but to keep the figure more understandable, this has been left out. In the earlier emergency messaging use cases, the information retrieved from the content providers was generated by the providers themselves, but in the case of content analysis, the retrieved data is often created by the clients – the same clients that use the front-end service. In fact, only metadata is synchronized between the front end and content providers, while the actual data content remains on the content provider service and is only temporarily accessed when needed by the back ends for analysis purposes. The metadata varies for each content provider and data type: for photos it contains the photo URL and perhaps Exif information or annotations the user has associated with the photos; for social media profiles, the metadata could contain the messages the user has posted to his/her account.

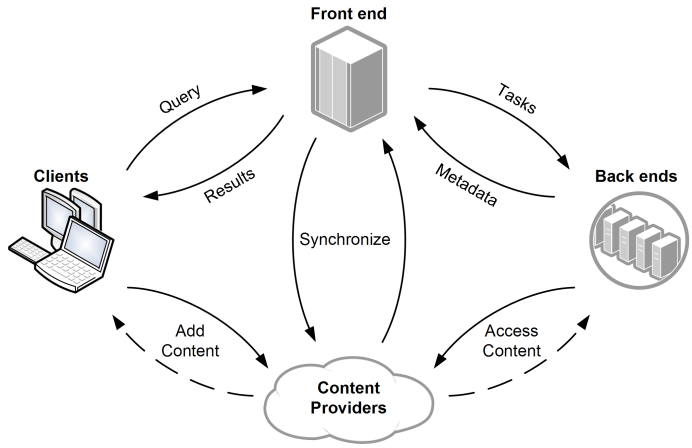

Figure 4.4: High level presentation of the generic architecture from Publication IV and Publication VII.

Generally, there are three key issues that dictate which data – or metadata – is retrieved during the content provider synchronization.

Firstly, the use case creates the basic requirements of what is required and what can be omitted. Secondly, the technical limitations regarding certain types of content can cause problems. For example, in the case of photos or videos, there might be extensive storage space and network bandwidth requirements, and in practice, it might be difficult or even redundant to duplicate the data on the front-end service, and it might be better to simply synchronize a minimal set of metadata containing information on how to access the data. This would leave most of the heavy lifting for the content provider, freeing resources from the front-end service. It is also worth noting that, after the initial analysis phase, the

source data is not necessarily needed and queries can be executed based on the generated analysis results. The third important point is not necessarily relevant for research or testing purposes, but is crucial for the final end-user product: the legal restrictions. It is entirely possible that the content provider terms of use or intellectual property laws limit the use of data – storing and retrieving metadata or temporarily caching data for analysis purposes is usually considered fair use, but making persistent copies of data might not be.

The architecture does not consider the physical construction of the working system, but the options for enabling better scalability are discussed later in the subsection 4.2.2. The architecture does not define where the individual participants should be located in relation to each other. The content providers are most likely located separately from the front end and back ends because of their nature as external services. The front end and back ends can be located in a single cloud environment or in their own individual environments, which are in turn connected through the public Internet, Virtual Private Network (VPN), Demilitarized zone (DMZ), or similar network structure. From the client's perspective, the front end and back ends appear as a single service. However, because of the way the interfaces are implemented, each back end could also be considered to represent an individual service, and can be directly accessed by clients and other services. On a higher level, the architecture can be regarded as being an implementation of SOA, and the design takes many features from SOA design, such as unassociated, loosely coupled and self-contained services and distributed capabilities.

In principle, the participants (front end and back ends in particular) could even be located on the same physical hardware (server) and still implement the presented architecture although it has been primarily designed for interfacing systems, which are too complex to be implemented on a single hardware instance. By design, the back ends could be duplicates (for scalability) or simply offer different capabilities (for example, one back end could provide photo analysis, while another would offer text summarization).

4.1.5 Documentation

All of the presented iterations utilize some level of automation in the process of providing documentation for the architecture. In the first (subsection 4.1.1), second (subsection 4.1.2), and third iterations (subsection 4.1.3), the automated process was mainly limited to utilizing Javadoc, Doxygen, or a similar tool for producing method descriptions based on source code comments and annotations, which is a very typical way of providing documentation. The final documentation, including the usage and data structure examples, was created manually and consisted of Word documents or Portable Document Format (PDF) files. In the first three iterations, the development work was mainly done by one internal team, which had full access to the source code and other material related to the development process. In practice, this meant that the production of the final documentation was of lower priority, and more specifically, the usage examples could be provided after the development process had finished.

During the development of the final iteration (subsection 4.1.4), the number of external teams increased, the access to source code and material became limited, and the importance of the designed interfaces grew. This also increased the importance of the API documentation and it became crucial to provide usage examples for the work-in-progress interface revisions. As the interfaces and data structures were occasionally modified based on the feedback from teams responsible for the back-end development, keeping the documentation and example code up-to-date became a real maintenance nightmare.

Modifying the Word documents and PDF files whilst maintaining a coherent look and feel to all the documentation can be a very tedious and error-prone process.

On top of the increased documentation requirements from external parties, the later iterations utilized a certain amount of code and design solutions from the earlier versions. The previous iterations were – depending on the project in question – either adequate or lacking in documentation. In particular, projects primarily oriented for research or prototyping contained code with poor or non-existent documentation or the documentation on the rationale of the design was missing, making it challenging to decipher the reasoning behind certain design choices (particularly, choices in implementation and source code).

These reasons led to the realization that the documentation process should be more tightly integrated into the development process and also into the API design itself [Publication VI], as it can be very difficult to modify the code style of a large codebase to adhere to different documentation methods later in development. In our case, the complexity and size of the final proof-of-concept implementation made it challenging to organize resources for testing multiple documentation strategies. Furthermore, as the proofs-of-concept were largely developed as part of research projects, which had their own documentation and reporting requirements, the need for easy-to-use and versatile documentation tools and methods was more acute. The parts of the documentation that could not be created through automated means (for example, by utilizing methods described in [Publication VI) were created using online tools. Tools such as Asana (task and project management service [Asana, 2017]) and The Bug Genie (bug and issue tracking [The Bug Genie, 2017]) were commonly used, together with online diagram tools such as Web Sequence Diagrams [Web Sequence Diagrams, 2017]. In fact, quick sketching of diagrams was often done with online tools even when the final diagram would later be done with a more heavyweight application like Microsoft Visio or a Unified Modeling Language (UML) editor. Ultimately, all documentation – with the exception of certain project reports that still required a traditional word processor – was created by web-based tools and submitted either to the task management service (which was limited only to project members) or published on the public wiki page [TUT Pori, 2015g]. The advantage of this approach was not only the relatively easy setup for the required tools, but also the fact that all of the tools also provided either an entirely free service for trial or limited use, or an open source version of the software.

4.2 Quality Attributes

The quality attributes – or non-functional requirements – are an important part of any architecture design [Chen et al., 2013; Cleland-Huang et al., 2013], and it is commonly accepted that the quality of a software system can be determined and evaluated at the architecture level with quality attributes [Bass et al., 2012; Matinlassi, 2006; Tarvainen, 2007]. For an architecture or a model to be valid, it must fulfill a set of predefined quality attributes, although in related research [Bosch and Molin, 1999; Dobrica and Niemelä, 2002], it has also been noted that the quality attributes of the final system cannot reliably be measured by simply looking at the software architecture design. The quality attributes can also be conflicting and interrelated with each other [Henningsson and Wohlin, 2002]. Keeping these points in mind, section 4.6 will show the applicability of architecture for final system design through the descriptions of proof-of-concept implementations, while this section concentrates on the attributes and their relations. Nevertheless, considering the importance of the quality attributes, and the fact that the research in this thesis

attempts to find out whether a generic architecture for the content analysis domain can be realized, it is crucial to take a look at the attributes that define the pre-requirements for overall architectural design in the content analysis use case.

Throughout the design and evolution of the generic architecture, certain characteristics were repeatedly focused on and deemed important based on observations and the experiences gained during the development, as well as, based on the performed background research. These characteristics – i.e., requirements or quality attributes – were as follows: interoperability, extensibility, and flexibility; scalability and availability; testability, usability, and maintainability; portability, adaptability, and reliability; and security. There are other quality attributes related to software development in general, and this section does not attempt to provide a complete, exhaustive look at software quality management, but instead attempts to shed light on the importance of the chosen attributes. Why the attributes were chosen, and how their fulfillment is guaranteed in the design of the generic architecture, are explained below.

4.2.1 Interoperability, Extensibility, and Flexibility

Interoperability refers to the ability of systems to usefully exchange information [Bass et al., 2012; ISO/IEC, 2011a; Kosanke, 2006]. Extensibility is the ability to acquire new features [Dobrica and Niemelä, 2002]. Flexibility is the degree to which a product or system can be used with effectiveness, efficiency, freedom from risk, and satisfaction in contexts beyond those initially specified in the requirements [ISO/IEC, 2011a]. The attributes are very closely related in the generic architecture. On the one hand, we need to make sure the various systems can understand each other, and on the other hand, we need to ensure that the chosen method of communication is flexible enough to allow a relatively easy way to add new features. The most important interfaces to consider in this respect are the one between the front end and the clients, and the one between the front end and the back ends – for the reason that there is very limited control over the interface towards content providers. The interfaces are illustrated in Figure 4.5 by *Interface A* and *Interface B*, respectively. Additionally, the construction of the API is further explained in section 4.3.

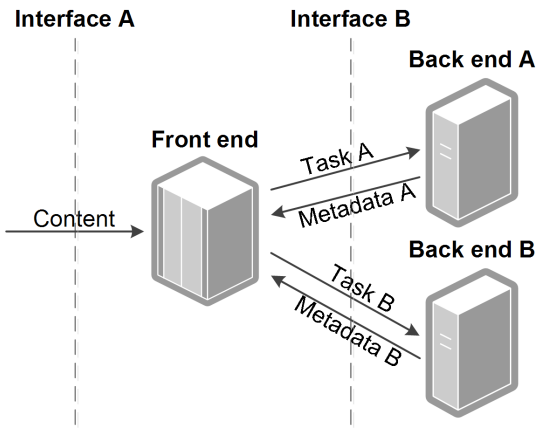

Figure 4.5: Realization of interoperability, extensibility and flexibility through task-based content delivery and analysis.

Figure 4.5 illustrates the content analysis side. Additionally, a functional system would most likely include query and possibly feedback methods on the client side (*Interface A*). These have been omitted because it generally depends on the use case which features and methods are exposed on *Interface A*. For example, the service could analyze the user's social media profile, including photos and status messages, but it might be only required to create a method that provides a simple list of keywords extracted, regardless of whether the keywords originated from the photos or from the status messages. In a way, *Interface A* is perhaps the simpler of the two interfaces. Based on the practical experience gained throughout the architecture development, *Interface A* should utilize the same data formats and types (if possible) as *Interface B*, but hide the functionality of *Interface B*. It is possible to pass parameters through *Interface A* that control the behavior of the methods of *Interface B*, but that might create unnecessary complexity to the overall design. When designing *Interface A*, one should think of the minimal set of features the clients really need and pursue that direction.

The *Content* in Figure 4.5 is a very generalized representation of the content delivery and in this case may contain data directly uploaded by the user or data (or metadata) synchronized from an external content provider, and can be of any type and size. Regardless of the origin of the content, the next step is to decide what to do with the content, which affects or even creates our interoperability and extensibility problem. Unfortunately, because of the inherit complexity of the analysis methods, it would be very difficult to implement all of the desired features on a single back end, or on the front-end service. On the other hand, the architecture does not enforce the implementation of a certain feature on any one component. To preserve better extensibility and interoperability (as well as maintainability), the roles of the components should be clearly defined. In the architecture, the primary purpose of a back end is to provide analysis capabilities, and thus, implementing analysis functionality on the front end is not recommended. In some cases there may be exceptions to this rule. For example, content analysis can be performed on the client if the client resources are sufficient and the results can be included as metadata in the content delivery. A practical example of client-based content analysis would be a camera that contains dedicated software or hardware for face recognition.

The solution offered for the problem is the RPC or hybrid REST Task-based approach presented in Publication IV in combination with capability-categorized back ends. In practice this means that the front end keeps a register of known back ends and their capabilities and based on these capabilities, [1] the service knows how to divide the incoming content into separate tasks and to which back end each task should be delivered. By utilizing multiple analysis back ends, as wide a range of metadata as possible can be gathered and more accurate search results can be achieved by combining the metadata [Publication IV][Sillberg et al., 2013].

The task-based approach can be seen in Figure 4.5. When the generic *Content* is identified as containing two separate data types, two tasks (*Task A* and *Task B*) are generated, one for each back end (*Back End A* and *Back End B*). Depending on the case and front-end configuration, it is also possible to send the same *Task A* to *Back End B* if B is capable of processing it. If the service needs to be extended with new content types, it is enough to create new capabilities, which in a properly designed system would have no effect on the existing features.

[1] An example of capability listing can be found in the proof-of-concept service implementation VisualLabel [TUT Pori, 2015c].

As mentioned before, the structure of the tasks can vary considerably, but there are a few general guidelines or issues that should be kept in mind when designing the content delivery [2]:

- Utilization of identifiers. The tasks should always contain a universally unique task identifier, and possibly a back-end identifier. Delivering the back-end identifier (and other relevant identifiers [Publication IV]) with the tasks releases the back ends from the responsibility of keeping track of their own identifiers per front end, because their own identifier is always included in each task. This also has another advantage, which may not be obvious. As long as the tasks are self-contained – i.e., they include all details that the back end requires to respond to the task – multiple *different* front-end services can utilize the same back ends, assuming that the network configuration and the authentication schemes allow it.
- Back end specific task contents. Creating the tasks individually for each back end can save bandwidth and reduce the need to transfer data that the back end is not going to use anyway, but will require a method for filtering included content [3]. Moreover, often the query methods in Interface A (client side) should also include functionality to filter the result set and similar features can be relatively easily added to tasks.
- How much of the task is of generic content and what is data type specific? The level of generalization should also be reflected in the design of the task format. The levels of data are more thoroughly discussed in section 4.5. The input and output data types and formats should be uniformly defined.
- The tasks should contain the requirements for what is to be processed and what should be the expected outcome, but the front end should not care about how the results are produced. In other words, the tasks represent the interface (agreement) between the front end and the back end, and define the inputs and outputs of the operation.

Regardless of the implementation of the task delivery, the back ends will return metadata containing the generated results, which the front end will incorporate in its database. In situations where there are no results, it is preferable to require the back ends to respond in any case, even if there are no actual errors present, as this will help the front end to manage the task scheduling. If nothing is returned, it is difficult to figure out whether the analysis failed and requires future re-scheduling or if there simply were no results. Depending on the case, it might also be advantageous to keep track of the status of the analysis, for example, by allowing the back ends to return results for only part of the given task in cases where the full analysis would take a very long time.

4.2.2 Scalability and Availability

Scalability is the ability of a system to either handle increases in load without impact on the performance of the system, or the ability to be readily enlarged [Microsoft Developer Network, 2016], while availability is the degree to which a system, product, or component

[2] Example task can be seen in the VisualLabel proof-of-concept implementation [TUT Pori, 2015f].

[3] One possibility is to use separate data groups as in the VisualLabel proof-of-concept implementation [TUT Pori, 2015e].

is operational and accessible when required for use [ISO/IEC, 2011a]. The improvement of scalability was one of the two primary reasons for separating the front end and the back end, with the other reason being testability, which is discussed in subsection 4.2.3), and both topics are further explored in [Publication IV].

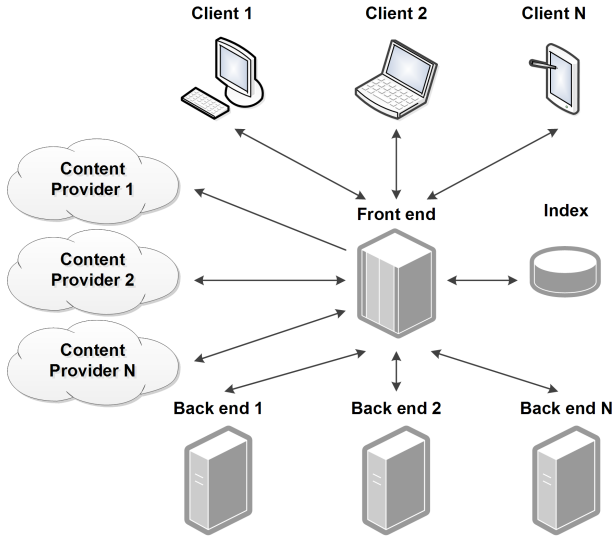

Figure 4.6: Scalability with a single front end and multiple back ends.

Figure 4.6 illustrates the basic use with a single *Front end* (and *Index*), multiple *Content providers*, multiple *Clients*, and multiple *Back ends*. To make the figure (and all subsequent figures in this subsection) easier to read, the communication between the *Front end*, *Back ends* and *Content Providers* is simplified, but the synchronization is assumed to take place as presented previously in Figure 4.4 in subsection 4.1.4. Each of the *Back ends* (and their nodes, as presented next in Figure 4.7) may also contain internal databases used for back end specific functionality (such as for storing feature vectors or learning data for photo analysis), but for clarity, these are not illustrated in the figures.

In Figure 4.6, the *Front end* is solely responsible for dividing the tasks between the many *Back ends*, either for load-balancing reasons or because of the (analysis) capability limitations of the individual *Back ends*. Depending on the point of view, the centralized control of balancing can either be an advantage or a disadvantage. The advantage is that the functionality can be entirely implemented on the front end, although that creates additional complexity and overhead in the form of keeping track of the current load status of the (analysis) *Back ends*. When designing the proof-of-concept implementations (subsection 4.6), the front end was never used for load balancing, but only for distributing the tasks based on the back-end capabilities.

Often the analysis back ends are complex systems themselves, and by default contain their own load-balancing or load-distributing models. This was also the case with the back ends used as examples in this thesis. The back ends are better aware of their own load status than the front end, and creating an elaborate balancing functionality on the front end would be both time-consuming and redundant. A more feasible approach would be to specify a task prioritization feature, which would allow the back ends to reschedule – or temporarily interrupt – on-going tasks for the duration of the analysis of more important

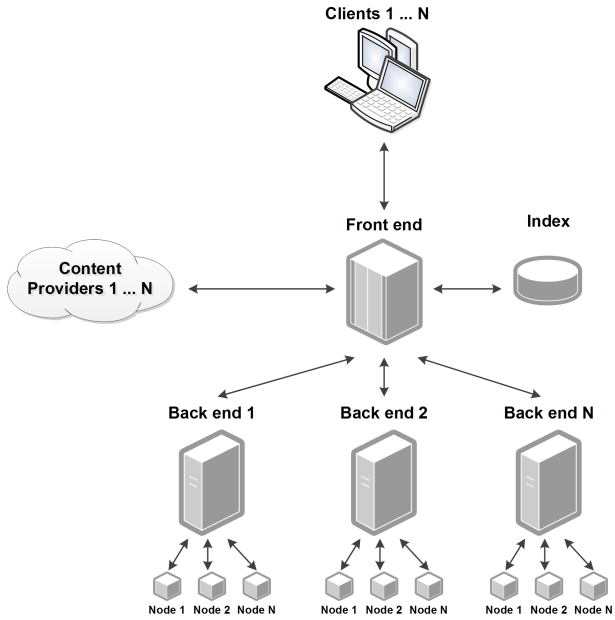

Figure 4.7: Scalability with a front-end service and multiple back-end services with multiple nodes.

tasks. The multiple *Back ends*, each with their own set of child *Nodes* used for load balancing are illustrated in Figure 4.7.

The computing resource requirements for analysis can be greater than the computing requirements of the *Front end* (service), but the tasks seldom require real-time processing. In fact, due to the technical limitations of the available processing power, the real-time analysis of large data sources may not be achievable in a practical scenario. On the other hand, client-side operations, such as content queries, should be answered within a reasonable time frame. High latencies can have a considerable effect on the user experience, thus creating a more acute strain on the front-end service performance. For this reason, it might be advantageous to multiply the front-end service, and it might be even more crucial for the end-user experience than the relative speed of the content analysis. After the initial analysis phase, the results indexed on the front end can also be used for queries, both reducing load on the back end and increasing computing requirements for the front end. Thus, from the perspective of the end users, the key factor in guaranteeing good availability lies more on the functionality of the front end than in the operation of the back ends. Delayed task execution on back ends generally results in an iterative, delayed delivery of results, but problems on the front end will cause immediate and noticeable effects on the use of the entire system.

The *Front end* instances are not interrelated and can be duplicated, as illustrated in Figure 4.8. The *Front ends* can freely create new tasks as long as the uniqueness of the task identifiers can be guaranteed, and because the tasks contain the required identifiers and callback URLs, the *Back ends* can easily respond to tasks that arrive from any *Front end* instance. In practice, the design and implementation of the metadata index largely controls how the *Front ends* can be duplicated, being the sole storage of the task-, user-, and configuration-related information. Several free and/or open solutions exist

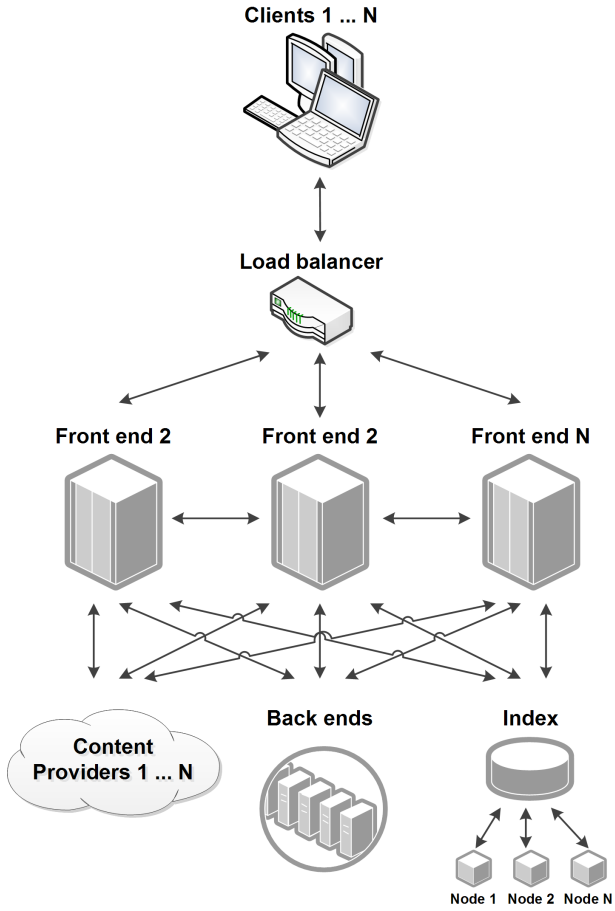

Figure 4.8: Scalability with multiple front ends and multiple back ends.

for managing distributed databases (for example, [Apache Software Foundation, 2016c,d; Oracle Corporation, 2016a]). The topic of metadata indexing is also briefly discussed in the author's own publication Publication V, as well as in a more general fashion by other authors [Lourenço et al., 2015].

The pattern shown in Figure 4.8 can easily be combined with back end specific nodes, as seen previously in Figure 4.7 without much effect on the generic architectural design. Which approach to take depends on the use case: the higher the expected usage (load) on the service, the more complex (scaled) the front end and back end duplication needs to be. The performance of each individual component – both system and programming components – has a significant effect on the performance of the complete system, making it difficult to recommend any particular pattern. Thus, this subsection will settle for describing the options and will leave the choice to the actual developers of the final systems.

4.2.3 Testability, Usability, and Maintainability

Testability is the degree of effectiveness and efficiency with which test criteria can be established for a system, product, or component, and tests can be performed to determine whether those criteria have been met [ISO/IEC, 2011a]. Usability is the degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use [ISO/IEC, 2011a]. Maintainability is the degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers [ISO/IEC, 2011a].

To achieve better testability, usability, and maintainability, the generic architecture follows the design principle of Separation of Concerns (SoC) [Dijkstra, 1982; Reade, 1989]. The features are encapsulated into separate components (front end, back end, client, content provider) and by defining relatively loosely coupled interfaces, each component can be individually tested for both performance and usability [Publication III]. Using REST or REST like methods in the service design, each part can be easily tested either directly by using a standard web browser or indirectly using an HTTP client with a slightly larger feature set [4].

The usability of individual components of the architectural design, and the design of the connecting interfaces, can be achieved by taking advantage of requirements engineering (such as defining use cases) and further refinement through auditing within the project group [5]. To validate the usability of the content analysis concept, and thus the feasibility of the architecture, user trials were performed, although these concentrated mostly on the design and implementation from the perspective of the web client [Malik and Nieminen, 2014; Sillberg et al., 2013] – or the user's perspective – with the teams responsible for the development of their respective back ends performing their own benchmarks [Publication VII]. From the user's perspective, it might appear that the quality of the results provided by the analysis back ends would play a crucial role in the usability of the system, but this is only partially true. From the architecture's perspective, the quality of the results is irrelevant. There are two primary reasons for this:

- The architecture is simply the generic transport that enables the basic functionalities. It defines the inputs and outputs, the requirements and desired outcomes, but the actual data content – both source data and analysis results – can be anything within the boundaries of the designed specification. In an ideal case, to preserve generalizability and extensibility, the architecture – unlike the implementation or the analysis algorithms – should not have any opinion on the data.
- Analysis results are seldom perfect, and a more important feature is one that allows real-time or off-line improvement and learning in order to achieve better results in the future. This kind of feature can be implemented by utilizing both direct and indirect observation of the user's actions, and the gathered information can be relayed to the analysis back ends by using the same task-based approach as utilized in the delivery of the analysis tasks [Publication IV].

It is a well-known fact that software maintenance can be expensive [Dubey and Rana, 2011]. One of the primary costs lies in software enhancement and in the addition of new

[4] It is also relatively simple to create a web-based test page for running basic tests for HTTP operations [TUT Pori, 2016b].

[5] The use cases for the VisualLabel proof-of-concept implementation are available at [TUT Pori, 2015d], and are also discussed in Publication VII

features [Glass, 2002], with the most important maintenance artifact being the program code [de Souza et al., 2005]. To enhance the usability and re-usability of an existing code base, proper documentation is crucial, and in many cases this can be achieved, if not entirely, then at least assisted by utilizing automated tools [Publication VI]. In the generic architecture presented in this thesis, the automatic generation of documentation is taken into account in the interface design, and is more thoroughly discussed in section 4.3, and also in Publication VI.

4.2.4 Portability, Adaptability, and Reliability

Portability is the degree of effectiveness and efficiency with which a system, product, or component can be transferred from one hardware instance, software, or other operational or usage environment to another [ISO/IEC, 2011a]. Adaptability is the capability of the software product to be adapted for different specified environments [ISO/IEC, 2011a; Tarvainen, 2007]. Reliability is the ability of a system to remain operational over time [ISO/IEC, 2011a; Microsoft Developer Network, 2016]. Reliability depends on individual components, component interactions, and the execution environment, and not all the information related to the reliability schematics is always available when designing an architecture [Immonen and Niemelä, 2008]. Thus, reliability is not strictly an attribute of the design, but more of the implementation, but in this case the means chosen to improve adaptability and portability can equally well be suited for reliability.

In the design of each iteration of the architecture (section 4.1), and when implementing the proofs-of-concept (section 4.6, one of the main goals has been to use commonly used, free, and open components in combination with standard or industry de facto protocols. Much of the basic features required for today's systems have already been implemented in various publicly available libraries or frameworks, making it redundant to start the development from scratch. Depending on the developer's know-how, custom designs and implementations can also create issues already solved by others.

It can be challenging to find pre-made solutions for every content analysis algorithm, nor are there always full featured protocol implementations available to take advantage of, but below the content analysis layer, the web services, servers, and communication methods are not that different from their more traditional web counterparts [Publication III]. The advantage of utilizing well-known and tested components is also reflected in portability, as many solutions (for example, the Java environment) are by design portable from one environment to another.

4.2.5 Security

Security is the degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization [ISO/IEC, 2011a]. There has been a lot of research on the security of software and networked systems [Gilliam et al., 2003; McGraw, 2006; Microsoft Corporation, 2003; Wang and Wang, 2003]. The generic guidelines presented in the related research are in many cases applicable to the domain of this thesis (as the work *is* related to both of these topics), but in this subsection more attention is given to the specific issues related to the content analysis environment and the generic architecture.

Figure 4.9 presents the same (final) iteration of the generic architecture as in Figure 4.4 in section 4.1.4 with the addition of the illustration of the flow of data through the system. A lot of personal data can pass through the system, which means that the minimum

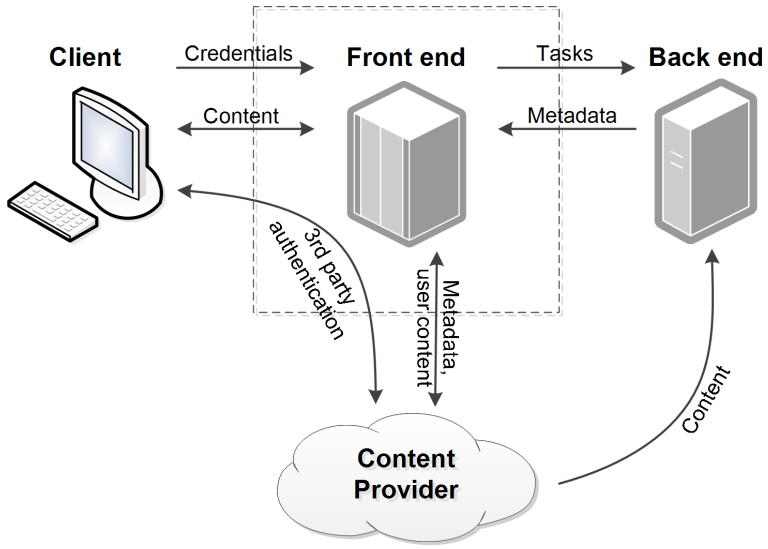

Figure 4.9: The flow of data throughout the system.

security measure to take is to use encrypted communications, by utilizing Secure Sockets Layer / Transport Layer Security (SSL/TLS). Some of the data can easily be understood to be confidential, such as the credentials provided by the clients upon authentication, or the data that passes through the front-end service when performing a third-party authentication using an external identity or content provider [Rantanen et al., 2017].

Some of the data may or may not require a more secure approach. The content exchanged between the client and the front end, and the metadata and user content synchronized between the front end and the content provider, can in some use cases contain sensitive information about the users of the system. If the data is something readily available on the Internet, the security requirements may not be so high, but if transferring, for example, a user's private social media profile, a more subtle approach is in order. In practice, there is very rarely any real need to implement different security policies for different metadata or user content, and usually a certain security practice is either directly dictated or at the very least recommended by the content provider – and there are generally no good reasons not to follow the provider's recommendations.

Similarly, the tasks and the metadata generated by the back end could contain the user's confidential information, although user passwords or other credentials should never be passed to the back ends. As a more general rule, no single party should have any credentials or any other personal information about the user that they do not absolutely need. If the back ends require access to private data located on an external content provider, a better approach than giving the user's credentials is to re-route the back-end requests through the front end, which will provide static links to the content [Publication IV]. Depending on the available network topology, one option for enhancing security could be the utilization of a DMZ or VPN solution for the front end and back end communication (content providers usually cannot be included because of their external nature), but even in this case, there is usually no valid reason to give any user credentials to the back ends. The fewer places the user's personal details are located, the lower the risk that they are leaked, stolen, or otherwise misused.

A case of particular interest is the content directly or indirectly retrieved by the back end from the content provider. As mentioned, no credentials are provided, but depending on the case (for example, using the social media profile case), the content may contain a lot of details about the user and it is not completely out of the question that the identity of the user (or other users for that matter) could be discovered by simply analyzing the provided content. Another issue closely related to the concept of security is retaining the user's trust. From the purely technical perspective, a lot of user details from hobbies to friends and even to daily habits can be extracted through the means of content analysis, but in principle, only information that the user has given consent to use should be processed and indexed. In principle, this is a point that is just as important to bear in mind as the utilization of transport layer security, but one that is far easier to forget in practice.

4.3 API Model

The design of interfaces is perhaps the most important point in the description of an architecture. Thus, the high-level architecture is presented in Figure 4.10 from the perspective of the interfaces. The diagram is a combination of different diagrams originally presented in Publication VI and Publication VII. The structure and flow of the diagram is similar to that presented previously in Figure 4.4 in subsection 4.1.4 with clients on one end (Figure 4.4 left and Figure 4.10 top), and the back ends at the other end (Figure 4.4 right and Figure 4.10 bottom). If compared to Figure 4.5, the *Client APIs* are Interface A, and Interface B consists of *Server APIs* in combination with *Back end APIs*.

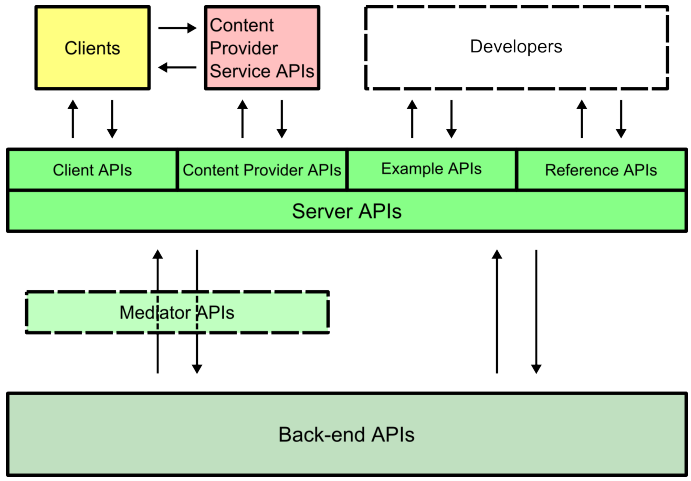

Figure 4.10: API model of the generic architecture.

In this case, *Clients* is a more abstract term, and the same clients (i.e., one application) do not necessarily access both the *Client APIs* and *Content Provider Service APIs*. Developers are a group (and not an interface or an end point – this is illustrated in the figure by the different, dotted outline of the *Developers* box) that develops, debugs, or designs applications and components for an implementation based on the architecture. What the *Content Provider Service APIs* contain depends on the use case, but they are often the interfaces that are utilized to gain access to the user's content or content metadata, as well as for performing third-party authentication when required.

The primary reason for placing *Clients*, *Content Provider Service APIs*, and *Developers* on the top of the figure – on the same level – is to enhance the visual clarity of the figure, but they also have one thing in common. In general, the three participants on the top have more limited access permission for the front-end service (presented as the large box with multiple APIs in the middle of the figure) than the back ends, which implement the *Back end APIs* or the optional *Mediator APIs*. Even in a scenario in which the back ends are not located in a secure network with the front ends, by nature they create metadata that is closely associated with the (personal) content of the user. An important issue to keep in mind is that, even when the user content is delivered to the back ends in an apparently anonymous manner, by studying the automatically generated metadata and the information extracted from the user content, it might be possible to discover the true identity of the user, or even identities of *other* users (for example, identities of the user's friends through face recognition performed on image content). If the back ends utilize previously analyzed content or analysis results from the user in question or from other users in future analysis, the chances of discovering user-related data increase, but even a single analysis task can contain enough data for the discovery of confidential details about a user.

In the case of the three parties on the top of the diagram, the access is limited to a very specific set of content: the private content they already possess or otherwise would have access to; the content publicly available; and in the case of content providers, to the metadata shared by the front end when accessing the *Content Provider Service APIs*. In other words, it is much easier to control what information is shared with *Clients*, *Content Provider APIs*, and *Developers*. On the other hand, the back ends are considered (more) internal in the service composition, and they should be more trusted in the way they operate with the user content, as it is much more difficult to control what the *Clients*, *Content Provider APIs*, and *Developers* do with the content they are provided with. Therefore, when designing the APIs, one should carefully consider how the user content is used, accessed, and shared amongst the participants of the content analysis environment.

Example APIs provide methods for generating example types and objects generally used in the system environment. *Reference APIs* provide functionally identical methods to *Client APIs*, *Server APIs*, and *Back end APIs*, or a functional subset of the three APIs [Publication VI]. They can also be exactly the same as their counterparts, but with access limited to the test data, or separate APIs with mockup implementations. Regardless of the implementation, their primary purpose remains the same, to allow testing without affecting the working behavior of the service. Today, many services offer public – or conditionally public through mandatory user registration – APIs, and it can create additional value to the developers if a "sandbox" environment can be provided for testing purposes. The disadvantage of creating dedicated – or dedicated through the use of mockup data or test users – testing methods is the increased extra work. The required development time varies depending on the complexity of the base service. The advantage, in addition to providing the developers with a more usable service, is the improvement of software management through the possibility of utilizing the methods in the automatic generation of documentation and example cases (as described in [Publication VI]). The *Example APIs* can also be used to generate objects usable in other system tests, such as in unit tests, and *Reference APIs* can be used to test the architectural concepts before implementing a fully working version.

Based on the inherent nature of the APIs, as well as the practical experience gained when designing the various iterations for the generic architecture, the following ground rules or

guidelines for each API can be described:

- *Content Provider Service APIs* and *Content Provider APIs*. The interface between these two is always defined by the content provider in question. The interface may or may not be optimal for the use case, but there is seldom any direct way to affect it. Sometimes the content providers offer pre-made libraries that implement the interface for ease of use – just like the interface, these may or may not be easily usable in the architectural design.
- *Client APIs*. These can be freely defined according to the preferences of the architecture designer – but with great power comes great responsibility. These APIs are also the primary external end point, and especially if the APIs are designed for public use, great care should be given to the design process. If these are poorly designed, the usability and desirability (whether the users and developers want to use the service or not) of the service can be significantly reduced.
- *Example APIs* and *Reference APIs*. These are optional to implement, but can offer added value to the service, and provide possibilities for documentation and maintenance practices – with the cost of increased development time.
- *Server APIs* and *Back end APIs* are the two end points of the same interface, the former being located on the front end and the latter being implemented by the back ends. This is perhaps the most important interface for a functional system. It should be generic and unified enough to allow ease of use and extensibility while at the same time enabling the various requirements of the multiple back ends. The structure of this interface is discussed in more depth in section 4.5.

One part missing from Figure 4.10 is one often found on services designed by following the traditional *web service* approach: an interface, which could be used to access and retrieve the web service descriptions using, for example, SOAP or WSDL. The model does not really recommend whether to include such an interface or not. *Client APIs*, *Server APIs*, and *Back end APIs* as the main interfaces could all have their own corresponding description interfaces. These interfaces, even though in some cases they could be crucial for the overall architectural functionality, are not always relevant for the basic use case scenarios – and in practice, were never required when implementing our proofs-of-concept. Additionally, the common interface description languages (such as WSDL) present the interface, (i.e., the inputs and outputs), but it is up to the programmer to design and implement the logic behind the interface. This, of course is the purpose of the description languages, but ultimately it does not remove the need to read the documentation, use cases, and examples to fully understand how a service works. In the past there was a lack of easy-to-use frameworks and software platforms that could be utilized to implement web services, and so the creation of the interfaces was a laborious task. The automatic stub generation offered by many WSDL style approaches provided some level of automation. The disadvantage was that the service descriptions themselves were rather complex, and for an experienced programmer it was often much easier to simply write the code for the services rather than to design the WSDL files. This kind of approach was perhaps not the “right thing” to do, but was nevertheless often the case. Today, many methods (such as annotation-based interface definitions) have made creating the interfaces much easier, further increasing the temptation to ignore the interface description languages. There is an on-going design trend, which favors simplicity in interface design in combination

with non-machine readable interface descriptions (for example, REST API with human readable online API documentation, a paradigm used by many large software companies. The relative increase of REST, RPC, and REST/RPC hybrid APIs compared to the amount of more traditional "web service" (e.g., WSDL) based solutions has also been observed by other authors [Maleshkova et al., 2010].

4.4 Interface Layer Model

This subsection compares the interface layers of the generic architecture with other commonly used layer models. The purpose of the comparison is not to prove the superiority of the generic architecture, or even to highlight the differences, but to provide a description of the important layers of the architecture and their positions compared with other existing models.

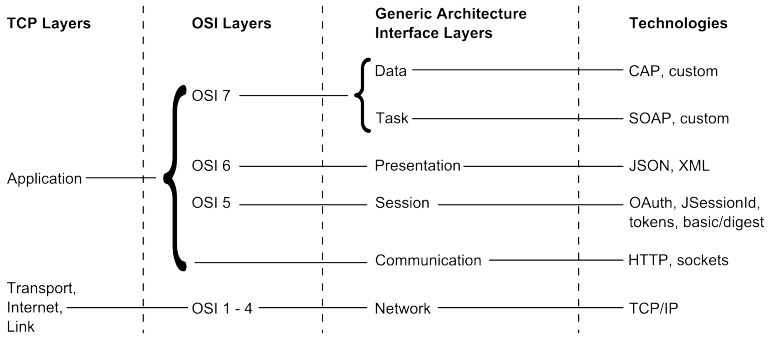

Figure 4.11: Interface layer model as compared to other models.

The generic architecture can be divided into six primary layers – data, task, presentation, session, communication, and network – of which the first five are of more importance for the context of this thesis, with the network layer implemented by the underlying network architecture. The layers are presented in Figure 4.11, and compared with the TCP/IP [IETF, 1989; Microsoft Technet, 2016] and Open Systems Interconnection (OSI) [ISO/IEC, 1994] layers. The figure also shows examples of the technologies used to implement the layers – the example protocols and practices are also the ones used in the proof-of-concept implementations (see section 4.6).

The TCP/IP model consists of four layers (*Application*, *Transport*, *Internet*, *Link*), of which the first layer equals OSI layers five through seven (Session, Presentation, Application) and the last three equal OSI layers one through four (Physical, Data link, Network, Transport). The layers of the generic architecture mirror the ones found on the OSI model because there is no valid reason to invent entirely new layers. Thus, the generic architecture layers make only two changes: the OSI application layer is separated into two layers (*Data* and *Task*), and one new layer (*Communication*) is added.

The re-layering of the OSI application layer is done to illustrate the existence of a more generic application layer, which in this thesis is called the *Task* (or operation, or workload "container") layer, and a more payload- or content-specific layer named *Data*. The architecture layers presented in the figure are mainly related to the interface between the front end and the back ends, but are equally applicable to illustrate the two other interfaces (i.e., front end–client and front end–content provider interfaces) with one exception. The Data/Task separation is specific to the communication directed towards

the back ends and does not exist on the other interfaces. In the case of the content providers, the application layer formats and practices vary between the providers in question. For clients, only the *Data* layer might be required – perhaps in combination with a use case specific data container format.

The *Communication* layer fits into a more generally defined *Application* layer of the TCP/IP model, but it does not have a well-defined position in the OSI model, being located somewhere between OSI layers one through four and five through seven. Moreover, even for the TCP/IP model, one could argue that the *Communication* level could belong to the *Transport* layer. The *Communication* layer is added to visualize the role of HTTP in particular. In the past, HTTP used to be more clearly defined as an *Application* layer protocol, but in the modern Internet it is more of a lower-level protocol utilized to "transport"[6] the higher-level application content. For this reason, the *Communication* layer could also be called the Transport layer, but in this context it is termed *Communication* (as it enables the higher-layer communication and content transfer) to separate it more clearly from the similarly named layers of the two other models. In a communication pattern, where HTTP or any comparable protocol is not used, the *Communication* can be thought to be missing or consist of the raw communication protocols utilized on the socket level.

Excluding the *Network* and *Session* layers, the other four layers (*Data*, *Task*, *Presentation*, and *Communication*) have their corresponding data layers, and these are further discussed in the following section 4.5. The *Session* layer cannot be neatly placed in any of the data layers, making its role as a layer slightly peculiar – both in the generic architecture layer model and in the OSI model. In principle, in any model, for a layer to be *a layer* it should be separate from the other layers as well as having a certain dependency towards the layers below and above it, but depending on the technologies used the authentication – or session creation – can take place on the *Network* layer (e.g., routing, DMZ), on the *Communication* layer (e.g., HTTP digest, basic authentication) or on the *Data* layer (e.g., custom implementations, identifiers). The data format of the authentication process can be anything from JSON to cookies to plain or unstructured text. Due to the layer's elusive nature, great care should be taken when designing it. In a well-designed system, the authentication – and authorization – schemes should be separated as cleanly as possible from the content delivery to allow future modifications and additions. Additionally, the popularity of third-party authentication has increased in recent years and support for it should be included in the overall design from the beginning, because ad hoc security implementations can cause potential problems [McGraw, 2002].

4.5 Data Layer Model

The common data layers for the front-end–back-end communication and content-based analysis in general, as utilized in the design of the generic architecture, and in the proof-of-concept implementations, can be seen in Table 4.1.

Starting from the bottom, the first layer is the *Communication* layer. In every iteration of the architecture (see subsection 4.1), this layer was implemented using a standard set of protocols because of the wide variety of existing libraries and frameworks. The protocols operating at this layer should (only) provide generic operations (create, retrieve, update, and delete) for the basic content transfer and communication needs and on top of these

[6] TCP/IP and OSI models both include a transport layer, but for both of the models this layer is a packet-switching or segmentation layer, and is not directly responsible for higher-level content delivery.

Table 4.1: Data layers and characteristics.

Layer	Characteristics	Protocols	Implementation
Data	Task type specific.	Extended XML, extended JSON.	Depends on task type, very difficult to generalize.
Task	Unified, typed, generalized.	Extended XML, extended JSON.	Custom defined, identical for all payload types, type information for detecting task types.
Representation	Standard, de facto.	XML, JSON.	Implemented by various libraries, frameworks and parsers, no modifications to default implementations.
Communication	Standard, de facto.	HTTP.	Implemented by various libraries and frameworks, no modifications to default implementations.

more advanced features can be built on. One of the most commonly utilized protocols of this level is the HTTP.

The *Representation* layer defines the generic structure used to represent the data content. In general, it is better to choose a commonly used format (XML and JSON being the most popular choices) to take advantage of existing solutions to both reduce the development time and to improve interoperability and extensibility. By designing a custom format, performance improvements can be gained, but designing and implementing an extensible format is no trivial task and such an approach might be limited to cases where the actual data to be transferred is simple or when good performance is crucial [Publication I][Publication III]. The two primary concerns for the overall performance on the *Representation* layer are the overhead caused by the data structure of the chosen format and the efficiency of the parser implementations. In the content analysis use case, both of these concerns have only a minimal impact on the overall performance because of the relatively long duration of the actual content analysis process.

In practice, it is often very difficult – if not impossible – to define a data format that is applicable for all intents and purposes. As illustrated in Table 4.1 above, the data is dependent on the task type, or in other words, of the operation in question. The actual data can only be generalized up to a point. For example, videos, audios, and photos can have an identifier, an URL, and a filename, but there are usually details that require content-specific fields or elements. For example, video and audio can have a bit rate, but photos cannot. Implementing all the required fields on a single type would create a bloated, hard-to-maintain super type, and a better approach is to have a simpler base type, which is extended for more complex use cases – an approach very common in object-oriented programming. The split into *Task* and *Data* layers follows from this same principle: the *Task* contains the general structure required for delivering and receiving analysis tasks and responses while the *Data* describes the task-specific data content.

Depending on the case, multiple tasks can utilize the same data types and the data types themselves can be extended from a more simple base specification. It is also worth noting that even though it can be difficult to generalize the *Data* layer, standardized formats for presenting individual data types (e.g., date and time formats and character encodings) exist, and should be used when they are able to increase interoperability. The back-end requirements, as well as the currently utilized data, should not entirely dictate the data formats – otherwise problems may arise when the specifications are to be expanded with

new use cases and operations. Thus, the design of the *Task* layer should be such that the content on the *Data* layer is irrelevant and could be changed without modifications on the *Task* layer. The *Task* layer can be thought to represent the generic container for data, and for content delivered from the front end to the back ends, the tasks usually consists of:

- General details of the task, such as the task and back-end identifiers, callback URLs, and the task type (e.g., video analysis task, social media summarization task...).
- Possible case-specific operations to be performed in pre-defined, numerical or textual, typed syntax. For example, should visual or audio analysis be performed for a video, or should face detection be performed for a photo?
- Accepted output data formats *if* required. In practice, assuming that one task type returns a certain pre-defined output, the data format can also be a valid approach.
- Additional data related to the task type, in a format defined by the *Data* layer specification. For example, a list of objects that define the accessible URLs for the files to be analyzed.

The tasks should only be delivered to back ends that can actually process them, based on the known capabilities of the back ends. The relevancy of the data from the perspective of the back ends can also be improved by implementing pre-defined back end specific data groups or filters in the task delivery mechanism. Whether filters are used or not, the back ends should never assume anything about the received data – i.e., all unknown content should be either ignored or responded to with an error code. Ignoring new or unknown fields or elements in the task and data contents can be used to allow the extendability of the format, but it can also cause potential issues. Based on practical experience in running the proof-of-concept implementations, the unknown data on the *Task* layer should not be ignored to lessen the possibility of errors in the analysis process. On the other hand, it is often safe to ignore unknown data on the *Data* level, assuming that the modifications on the data representation do not break backward compatibility (which they should not do).

In turn, the tasks – or task responses – returned from the back ends to the front end usually consist of:

- The same task and back-end identifiers as in the original task. The response can also mirror all the common fields of the original task, although this may be redundant, as based on the identifiers, the front end should be able to resolve all missing information.
- A status field, which can be used to either return the default OK message or an error code if the task processing failed. Note that, in general, back end specific error codes *should not* be allowed, and if for any reason additional information needs to be returned, it is much better to allow an optional message field for back end specific error codes used only for logging. Depending on the analysis methods used, the back ends can theoretically return an unlimited variety of errors, but the front end can only resolve – or even understand – a very generic set of errors (e.g., missing files, connection timeouts).
- The results in a valid format as described by the *Data* layer specification.

As can be seen from the nature of the *Task* layer, in some cases it cannot be used or its use is undesirable. Similarly to the interface layers presented in section 4.4, the *Task* layer cannot be enforced on the content providers if their interfaces do not by nature implement it. In the case of front end to client communication, there is necessarily no operation to define, and often only the data is required. Or to put it another way, the client queries are often simple enough to be managed with the combination of the basic operations supported by HTTP and a list of parameters provided either within the request body or in the query URL.

4.6 Proof-of-Concept Implementations

Based on the design science approach, a model or a theory can be validated by an iterative – or evolutionary – approach, and by rigorous testing and improvement of the chosen solution. In addition to the various architectural iterations, the proof-of-concept implementations, demos, and prototype systems can be thought to prove the feasibility of the presented architecture – if no practical systems can be built based on the architecture, what point would there be for the architecture to exist? For this reason, this section and its subsections will describe the implementations mentioned earlier in Table 3.1 in section 3.2, and also found in the publications included in this thesis. For each implementation, the original goals are presented, and the utilized technologies are listed. The technologies are listed only for the components that have been partially or fully designed by the author of this thesis, and for that reason the back-end technologies are not mentioned – but references to additional information are provided for each back end. Additionally, the relation of each implementation to an architectural iteration presented earlier in section 4.1 is described.

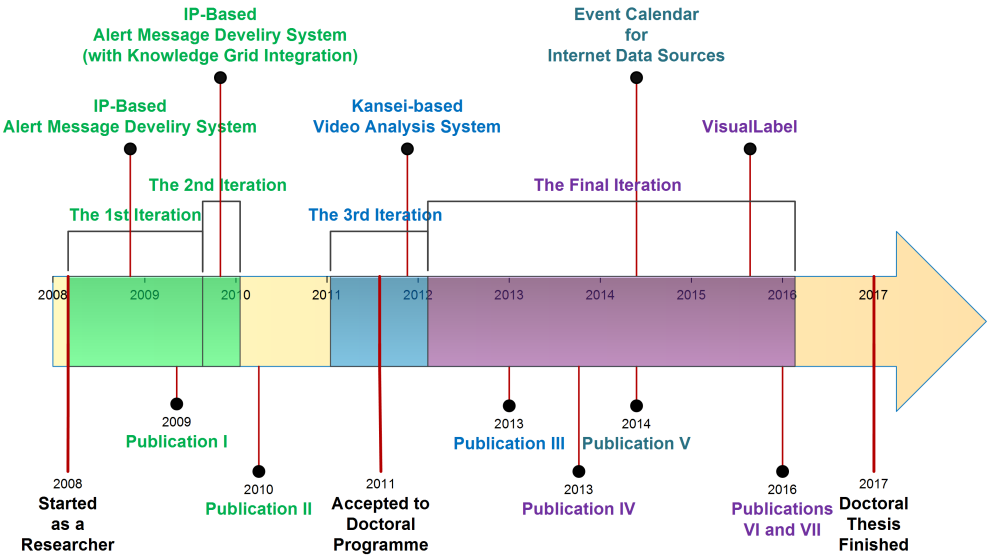

Figure 4.12: Visualization of the development process of the proof-of-concept implementations.

The relation between the implementations and the iterations of the architecture is visualized in Figure 4.12. The development time for each iteration can be seen in the figure, and the colors correspond to the projects the author of this thesis has participated

in, as illustrated earlier in Figure 3.1 (section 3.1). The proof-of-concept implementations were implemented as part of the research projects and not all of the implementations have an exact "release date," nor were all of the implementations made public beyond what was reported in the scientific articles. For example, source codes or detailed specifications (such as class diagrams) of the earlier implementations were only delivered to project partners and included in the final reports of the project, but they were not made publicly available. The dates roughly match the publication dates of the scientific articles, and the publications discuss the stable versions of the prototype implementations. The *VisualLabel* marker shows the date of the initial commit to the public source code repository located at GitHub [TUT Pori, 2015b]

4.6.1 IP-based Alert Message Delivery System

The first proof-of-concept system was an Internet Protocol (IP) based alert message delivery system, and it was based on the architectural iteration presented in subsection 4.1.1. The primary goal was to develop a system that could be interfaced with external systems for the retrieval of emergency messages. The messages would be delivered to clients by utilizing a SOAP [W3C, 2007a] based communication channel, which could also be used by the clients to send confirmation messages or further information about the emergency situation. Another method of delivery was Atom [IETF, 2005] based feeds, which could be accessed by any capable reader, simple web browser, or the implemented test client. The research focus was on the use of the standards or commonly used protocols of the emergency messaging field to enhance interoperability with other systems. [Publication I][Rantanen and Sillberg, 2009]

The key technologies used in the implementation were:

- For client – front end (server) communication, Common Alerting Protocol (CAP) encapsulated in SOAP for bi-directional communication, and Atom delivered using HTTP/REST were utilized.
- For front end – content provider communication, either the same SOAP-based approach or polling from external sources utilizing source-specific protocol (generally HTTP based retrieval) were used. Extensible Stylesheet Language Transformations (XSLT) was used to convert the retrieved messages to the client format.
- The front end was implemented in Java, using the Apache Axis2 [Apache Software Foundation, 2016b] framework for web service and SOAP functionality, and was deployed on Apache Tomcat[Apache Software Foundation, 2016e], and tested on Linux, OS X, and Windows Operating System (OS). The database was implemented on the popular MySQL server[Oracle Corporation, 2016b].
- The test client was implemented in Qt and C/C++, and tested on Maemo OS on the Nokia N810 device, and on Linux, OS X, and Windows OS.

As described in Publication II, the system was further extended to better utilize external back ends for content enrichment, and as a testing back end, the Knowledge Grid [Zettsu, 2012] from NICT was utilized. On the basis of the challenges faced and discoveries made, the architecture was updated to include a mediator component for message translation and the architecture evolved to its second iteration (presented in subsection 4.1.2).

4.6.2 Kansei-based Video Analysis System

The goal was to design and implement a system which could take advantage of the existing video analysis back end [Kitagawa et al., 2010] developed at Keio University Shonan Fujisawa Campus (SFC). The back end would utilize Kansei-based – or MMM based, to be more specific – analysis methods to discover the ”mood” or ”feeling” of the images. In practice, the back end would produce keywords applicable to each of the videos provided. These keywords could be used to enable search functionalities on the front end. The main technologies used in the implementation were:

- Client–front end and front end to back end communications utilized either XML or JSON based protocols extended with custom data format over HTTP/REST.
- The front end was implemented using Java and the Twonky [Lynx Technology, 2016] platform, and tested on Windows OS.
- The test client was implemented in Qt, and tested on Linux, OS X, and Windows OS.

The primary purpose was to test the feasibility of the video analysis by using video content commonly available on the public Internet, as explained in [Publication III], but by doing so, the usability of the larger architectural solution was also tested. As described previously in subsection 4.1.3, this third iteration saw a removal of certain redundant interfaces (namely, the removal of the SOAP query interface), which were no longer needed to realize the core functionality of the system.

4.6.3 VisualLabel

VisualLabel is the most complete implementation based on the generic architecture, and is based on the final architectural iteration described in subsection 4.1.4. The goal was to design a system which could take advantage of multiple analysis back ends, each specialized in their individual analysis methods, as well as utilizing the popular third-party content providers (Google, Twitter, Facebook) as data sources, content storages, and authentication providers. The metadata generated by the back ends (MUVIS [MUVIS, 2015], Summarizer [Forss et al., 2014], and PicSOM [Aalto University, Department of Computer Science, 2015]) would be used to realize an end user focused content management system for photo and video content. The system would also take advantage of a user feedback mechanism, which would enable the analysis back ends to learn and to improve their future results based on the user's actions. Publication VII provides the most conclusive description, but parts of the architecture and its components have been discussed in other publications: The usability from the user's (or client's) point of view has been discussed both in the publication co-authored by the author [Sillberg et al., 2013] and in a user trial performed and published by Aalto University, Finland [Malik and Nieminen, 2014], the overall architecture has been presented in Publication IV, and the integration with social media services and the utilization of text summarization has been described in another publication by the thesis author [Rantanen et al., 2017]. The technologies utilized in the implementation are:

- The client–front end and front end–back end communications use XML based protocol, with the back-end side generalized with a task-based container layer. The

data layer format in both cases is identical, although not standard because no generalized commonly used protocols for the content analysis domain exist.

- The front end was implemented in Java with the partial use of the dependency injection and security features of the Spring Framework [Pivotal Software, 2016], running on Apache Tomcat [Apache Software Foundation, 2016e]. The metadata index (database) was implemented by utilizing a database hybrid – a combination of MySQL server [Oracle Corporation, 2016b] and Apache Solr [Apache Software Foundation, 2016d], as described in Publication V.
- The web-based test clients were implemented in Hypertext Markup Language (HTML) and JavaScript (primarily with jQuery [The jQuery Foundation, 2016]) and tested on various desktop operating systems (Linux, OS X, Windows) and mobile devices (Android, Windows Phone, MeeGo, Sailfish OS, iOS).
- The service (consisting of the frond end and multiple back ends) was tested for scalability and portability in two configurations.
  - **In setup 1**, the front end and the summarizer back end were deployed on servers located in Pori, Finland. The MUVIS back end was deployed in Tampere, Finland and the PicSOM back end was running in Espoo, Finland. The communications took place over the public Internet.
  - **In setup 2**, the front end and all back ends were deployed on their individual virtual instances on the cloud service provided by Digile's Forge platform [Digile, 2015b].
  - **In both deployment setups** the actual deployment process could be achieved through automated tools (a combination of Apache Ant [Apache Software Foundation, 2016a] and Ansible scripts [Red Hat, Inc, 2016], as described in the VisualLabel Deployment Guide [TUT Pori, 2016c]).
- Additionally, the platform implementation utilized the standard Javadoc tool with customized Taglets to achieve automatic documentation generation for interfaces and classes with the possibility of creating method call examples directly within the interface documentation [Publication VI].

The source codes for VisualLabel have been published as an open source (Apache 2.0 license) and can be freely downloaded from GitHub [TUT Pori, 2015b]. The development guides [TUT Pori, 2016c] and use case descriptions [TUT Pori, 2015d], which can help to both utilize the VisualLabel proof-of-concept implementation in the future, as well as providing guidance for the design of similar or related content analysis systems, have been made publicly available. Unfortunately, the presentation material for VisualLabel on Digile's Forge cloud platform is no longer available, because the Forge Service Lab has been discontinued [Dimecc, 2016]. The feature videos are still available [Multimedia, 2016], but these only illustrate the back end and user interface features – and not the supporting architecture itself.

4.6.4 Event Calendar for Internet Data Sources

The Publication V presents an event calendar service that can be used to retrieve data from multiple sources (content providers) and present them either on a map or in a

more traditional calendar view – both views are part of a web client implemented using HTML5 and JavaScript. The service implementation is essentially the same as the one presented in subsection 4.6.3 with the exception of missing back ends. Content providers are present, but they offer a different set of data than in the content analysis case, and unlike the content analysis case, no back ends are needed as the retrieved data is simply indexed on the front end and sorting and presentation of the data is performed by the web client. Since the event calendar service shares the same codebase as VisualLabel, the implementation technologies for the front end and the database are the same (as listed in subsection 4.6.3 above).

In other words, the event calendar implementation can be seen to prove that the same basic approach utilized with the content analysis case in VisualLabel can be used at least on the calendar content as long as certain base rules – or similarities – are met: the system consists of a clearly defined front end service (a central point of the data exchange); and there are any number of external data sources or providers to be utilized.

Even though the use case presented in Publication V is not in the content analysis domain, the basic architecture would make it possible to add more advanced features easily through the introduction of content analysis back ends. For example, if you needed to extract more information concerning already retrieved events from the web, a back end could be dedicated to the task, or perhaps another back end could process the user's social media profile to obtain the user's personal events. Thus, the proof-of-concept implementation utilizes the same final iteration of the architecture explained previously in section 4.1.4 as VisualLabel, but because of a more simple use case, it does not take advantage of all of the features provided by the architectural design.

5 Conclusions

This chapter will present the main findings of the studies described in this thesis. The main focus of this chapter is on the summarization and presentation of the results of the studies (see chapter 4) described in this thesis in relation to the research problem presented in section 1.2. The relationship of the findings to the publications included in this thesis as well as the research problem has been visualized earlier in Table 3.1, and thus, the reader should refer to chapter 3 for further discussion on that topic. Chapter 3 also provides a more in-depth look at the research methodology utilized in this thesis.

The following sections will provide the answers to the research questions formulated in section 1.2, show the method of validation for the results of the study, offer an insight into future directions for research, and conclude with a brief summary.

5.1 Research Answers

Despite excessive research on the topic of content analysis, and on technology in particular, there is still a limited number of practical implementations. One of the problems is that most of the existing research focuses more on the design of the CBIR systems themselves, on the metrics or general principles of the analysis benchmarking, without taking into account the larger architecture required to implement a usable system. Additionally, in many cases, the various CBIR implementations are considered to be competing against each other, even though a more practical approach is to combine the systems under a single extendable generic architecture. There have been a lot of studies on software architecture on a more general level, and even though the researched architectures share similarities with content analysis systems, there has not been a comprehensive study on how the architectures should be used within the content analysis domain.

For an architectural design to be a valid software design paradigm and for it to allow the realization of a feasible system, the solution must comply with the quality attributes relevant for the content analysis use case – as reflected in the discussion in section 4.2. Additionally, the architecture must describe a reasonably high-level design and define the crucial interfaces required for the system to work as intended. Figure 5.1 presents the high-level architecture for the content analysis environment first shown in Figure 1.2 (section 1.1). The figure has been redrawn to include the API solutions recommended based on the studies presented in this thesis.

Figure 5.1 can also be used to illustrate the main findings of this study in relation to the research questions presented at the beginning of the study (section 1.2). The research questions and their answers are further discussed below.

Question 1: *What is the optimal method of communication for syntactically incompatible back ends?*

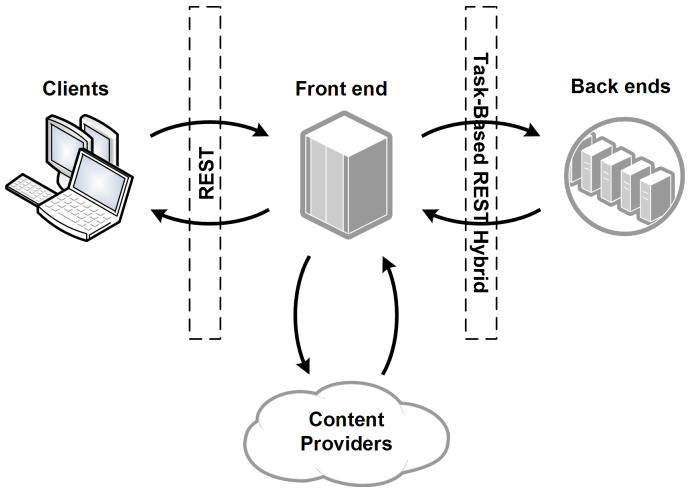

Figure 5.1: High level architecture diagram for content analysis environment with interfaces described.

Considering the interface between *Clients* and *Front end* in Figure 5.1, the optimal method of communication varies. In general, synchronized method calls can be easier to implement, but asynchronous communication can also be used if required. The REST style approach can be simpler to implement, can provide a more lightweight option for the implementation of the interface, and is usually easier for the developers of the system to learn how to use. In most cases, the *Front End* developers can dictate the entire specification, with the client developers simply following the specification. However, the interface specification should be as stable as possible with a minimal amount of changes that could break the backward compatibility.

The *Front End – Back Ends* communication is by nature of the analysis tasks asynchronous. The communication could be implemented with a "proper" web service design, though in the prototypes presented in this thesis, a more RPC like REST hybrid approach was chosen. The primary reasons for a more lightweight approach were the high initial learning curve and effort required for implementing a full web service. Additionally, the extra functionalities provided by web services (such as descriptive interfaces) were very rarely if ever needed in practice. From a developer's perspective, it would be easiest if both interfaces would utilize the same type of design, but in practice this might not be possible to implement – especially certain requirements imposed by the REST style could make the style very difficult to use for back end communication.

Despite the differences in the communication methods, both interfaces should utilize the same data structures and formats to simplify the design and implementation processes. The communication with the *Content Providers* (in Figure 5.1 above) is entirely dictated by the specifications given by the providers themselves. One of the formats utilized by the providers could be chosen for client and back-end side communication, although different providers generally use different or even mutually incompatible formats, making it difficult to use exactly the same formats throughout the entire content analysis environment. Also, even though the back-end requirements are an important part of the interface specification, ad hoc changes to the specification should never be taken based on the requirements of an individual back end, in order to preserve better interoperability and

compatibility with all back ends.

Question 2: *What are the crucial interfaces and data formats required for cross-platform communication?*

The higher-level interfaces are shown in Figure 5.1 and explained in the conclusions for *Question 1* above, and further discussed in sections 4.3 (Figure 4.10) and 4.5 (Table 4.1). In general, it is often better to utilize well-known (and thus, well-tested) protocols and formats, although in practice it might be difficult to find an existing generic solution that fits the use case at hand. Based on the experience of building the proof-of-concept implementations, it is difficult to recommend any specific data formats for the content analysis case. The de facto representation formats (JSON and XML) are extensively supported on most programming languages and on every commonly used platform, and picking either one will provide a stable base for further development.

To ensure interoperability and extendability, the chosen solution should provide a generic container structure (in this case, task-based containers), and the content-specific formats should be built on top of the container in order to minimize the relative "size" of the incompatible data structures in each request and response. There is very seldom a valid reason to utilize different formats in different communication scenarios (e.g., client – front end vs. front end – back end). In many cases, the various communications require only a subset of the whole (specified) data structure and these subsets should be well defined per use case in examples and documentation.

Question 3: *What kind of aspects are required to guarantee system maintainability?*

Ensuring good maintainability in a complex system can be a difficult task. The use of well-known formats and design patterns can reduce errors in the architectural design as well as in the design process itself. One of the most important tools for preserving the usability of the system for both designers and users (or developers) is proper documentation. Keeping the documentation up-to-date can be time-consuming, so utilizing automated methods for generating the appropriate documentation for all software products is preferred. It can be difficult to modify the documentation to adhere to a different documentation method or style later in the development cycle, and thus, how the documentation is to be created, utilized, and implemented in the APIs and in the source code should be a high priority consideration when designing any software architecture.

5.2 Validation

The research of this thesis follows the design-science paradigm. The research process was validated by following the seven guidelines [Hevner et al., 2004] created for understanding, executing, and evaluating design-science research, and by progressing through the steps of the mental model [Peffers et al., 2007] for utilizing design science. The relevance of the research has been shown by a study of the existing literature. The relevance has also been shown by the research efforts over multiple years of studies and research projects during which feedback, guidance, and future directions were received from educational and industrial partners working on the content analysis field. The search and development process for the architecture development was illustrated through the presentation of the architecture iterations. The validity of the individual iterations – as well as of the final architecture – has been enhanced by submitting the results to international peer-reviewed conferences and journals, which have provided feedback and improvement ideas for the research.

The participation of the author of this thesis in the development of the proof-of-concept implementations can be seen both as an advantage and disadvantage for the validation of the architecture. The participation proves the author's work and contribution for the design and development of the architecture, but it also means that no external party has independently realized a working system based on the design guidelines. In practice, the complexity and size of the content analysis system makes it challenging to organize an entirely separate team for constructing a proof-of-concept. On the other hand, the author's work in implementing the final proof-of-concept (VisualLabel) was minimal when compared to the implementation of the whole system. The back ends and test clients were implemented by other team members or by people working in different institutions or companies based on the designed specifications. All of the analysis engines utilized as back ends were products of multi-year research projects unrelated to the research on generic architecture presented in this thesis. Regardless of the unrelated origins of the components, using the generic architecture as a baseline, a working system was implemented and shown to work as intended. The implementation work took advantage of the material made publicly available online (development guides, use cases, data formats, and communication and API call examples).

Ultimately, the most important outcome of design-science research – in the context of information systems research – is the design artifact, in this case, the generic architecture itself. The realization of the architecture through instantiations (proof-of-concept demonstrations) can be seen to demonstrate both that the research problem has been solved and that the approach can be used to create a practical software implementation. The work presented in this thesis also described the crucial quality attributes required for a feasible content analysis system. The study of the quality attributes within the scope of the generic architecture was performed to validate that the solution presented can achieve its designated goals and purposes. The final iteration of the architecture, including the development guides, use cases, and the source code for a proof-of-concept implementation, has been published and is freely available for anyone. This can both help to validate the research, and also to provide a base for studies by other researchers.

5.3 Thesis Contribution

The studies presented in this thesis have been carried out over several research projects, all of which have targeted the same goal, namely how to construct relatively generic and practical interfaces that can be used to achieve the interoperability of complex systems. Keeping this in mind, the primary contribution of the work must be the answer to this research problem, which was first presented in section 1.2:

Is it possible to define a generic architecture, which describes how clients, back ends, and content providers can be connected in a meaningful way?

Based on the studies performed, this thesis claims that: yes, it is possible to define a generic architecture for the content analysis environment. More specifically, this thesis explained how the crucial interfaces for the system should be constructed utilizing the API model, presented in section 4.3 and the Data Layer Model, presented in section 4.5. Using the presented architecture, it is possible to separate the content analysis problem into abstract (layered) tasks, which can be flexibly delivered to analysis back ends for content analysis, search, and feedback purposes.

This thesis has also highlighted the lack of research on interface and architecture design in the content analysis domain, regardless of their importance for the realization of a

feasible system. Naturally, algorithms and analysis methods are the very essence of content analysis, but to increase the number of real-life systems, more thought should be put into how algorithms, content analysis systems, and methods can be utilized in an interoperable and flexible way to achieve a practical implementation.

Table 5.1: Contributions to the architecture structure and its primary components.

<table>
<tr><td colspan="2">Architecture
High-level architecture and interface specifications (Publication IV, Publication VII and sections 4.1.4 and 4.2.2, and Figure 5.1).

VisualLabel proof-of-concept implementation [Publication VII][TUT Pori, 2015g], source codes [TUT Pori, 2015b].</td><td rowspan="3">Documentation
Description of common quality attributes for a generic content analysis architecture (in section 4.2).

Documentation for the VisualLabel proof-of-concept (deployment guides [TUT Pori, 2016c], use cases [Publication VII][TUT Pori, 2015d], APIs and examples [TUT Pori, 2015a]).

Methods for automatic API documentation generation [Publication VI], source codes [TUT Pori, 2016a].</td></tr>
<tr><td colspan="2">APIs
API model (introduced in Publication VI and Publication VII, described in section 4.3).

APIs and examples [TUT Pori, 2015a].

Interface Layer Model (section 4.4).

Methods for automatic API documentation generation [Publication VI], source codes [TUT Pori, 2016a].</td></tr>
<tr><td>Communication Methods
Asynchronous transfer studies for the content analysis ecosystem [Publication II].</td><td>Data representations
Data Layer Model (section 4.5).

The task-based approach (described in Publication IV, further discussed in sections 4.2.1, 4.2.2, 4.2.3 and 4.2.5).</td></tr>
</table>

Furthermore, the contribution of the thesis to the research framework (introduced in section 2.1) and to the architecture structure (illustrated in Figure 2.1) is described in Table 5.1. The table lists the main outcomes of the research presented in this thesis and their relation to the architecture building blocks with references to the sections of this thesis or the related publications provided for the reader's convenience. Some of the work (namely, the implementation source code, wiki pages) listed in Table 5.1 has also been partially produced by other researchers. In addition, section 3.2 and Table 3.1 can be consulted for studies that have been conducted in co-operation with the author of this thesis, but are not included in the publications of this compilation thesis. All of the content should provide assistance for people who are involved in the design of systems in the content analysis domain.

5.4 Future Work

As presented in this thesis, the content analysis ecosystem and all of its components have been extensively studied. Utilizing the solution provided, there is no acute need to further explore the architectural design choices available for the content analysis domain. One possible future direction would be the design of a model for validating whether an existing implementation conforms to the guidelines and design decisions recommended in this thesis. The emphasis of the work presented here has been on the design and implementation of new systems although the provided guidelines can also be used when

making modifications to existing implementations. However, this work does not offer a precise and systematic means or a model for the validation of existing architectures.

The methods and technologies described in this thesis were briefly tested – through the implementation of proof-of-concept systems – in use cases not directly related to content analysis, but their applicability to a wider use has not been thoroughly validated. Thus, expansion to other domains is one possible future direction for related research. In particular, the improvement and design of methods utilized for automatic documentation might be an important future topic.

Another subject, not only related to the content analysis domain, but also to API design and usage in general is the rather strict "limitations" enforced by external APIs (such as those enforced by content provider APIs). There does not seem to be much research on how well the public APIs match common use cases – or the use cases as specified by the users of the APIs – despite the fact that the quantity of public APIs is growing [Anthony, 2016]. Additionally, even though many companies host online forums and bug trackers, the impact of the feedback is unknown – i.e., what is the effect of the feedback on the evolution of APIs in general? It is entirely possible that the way external APIs are implemented requires unwanted modifications to be made in the intended design of the system. Today, there are often many competing services available for commonly required operations (such as for image hosting), and based on experience of designing the proof-of-concept implementations – and from a developer's perspective in general – it would be interesting to see statistics or studies on how the commonly used public APIs evolve (the number of breaking and non-breaking changes, the average lifetime of an individual API, etc.), and how an individual developer (or team) can affect the development of public APIs.

Additionally, encouraged by the positive feedback received from the industry partners of the D2I program, the development of the VisualLabel framework and the proof-of-concept implementation will continue on the APILTA (Avoin PIlvipalvelukonsepti joukkoistetun Liikennedatan TArpeisiin, "Open Cloud Platform for Crowdsourcing Based Services", in English) project, which began at Tampere University of Technology, Pori Department in autumn 2016. The designed architecture and the task-based approach will be utilized for the collection, analysis, and refinement of traffic data produced by external back ends, mobile sensors, and crowdsourcing.

5.5 Summary

This thesis discussed the problems of designing and implementing a complex system for a content analysis ecosystem. The system – or architecture – would consist of any number of both desktop and mobile clients and of a single front-end service, which would utilize any number of external service providers and back ends designed for various content analysis tasks. These tasks could include, for example, text summarization, photo analysis, or content-based search. The background research suggested that despite the extensive studies on content analysis in general, there is a lack of research on how to implement a usable system consisting of all of the required parts, as the priority of existing studies has been almost entirely placed on algorithm and analysis method design.

The purpose of the research presented in this thesis was to study how to define a generic, extendable, and maintainable system within the content analysis domain. This thesis described the methods, technologies, and principles required in the architectural

design. Furthermore, the work was validated through several architectural iterations and proof-of-concept implementations, which were also presented in this thesis.

Based on the studies performed, it was concluded that it is possible to realize a generic architecture – or generic interfaces – by following layered API and data models presented in this thesis in combination with commonly used (standard or industry de facto) architectural and technical solutions (REST and hybrid REST/RPC) and representation formats (JSON and XML). In conclusion, the study shows that the existing technologies are sufficient for realizing the generic architecture when proper design of the interfacing protocols and formats is utilized.

References

Aalto University, Department of Computer Science (2015). PicSOM, The Content-Based Image and Information Retrieval Group. `http://research.ics.aalto.fi/cbir/`. Retrieved September 20, 2015.

Aihkisalo, T. and Paaso, T. (2012). Latencies of service invocation and processing of the rest and soap web service interfaces. In *proceedings of the 2012 IEEE Eighth World Congress on Services*, pages 100–107, Honolulu, Hawaii, USA. DOI: 10.1109/SERVICES.2012.55.

Aleti, A., Buhnova, B., Grunske, L., Koziolek, A., and Meedeniya, I. (2013). Software architecture optimization methods: A systematic literature review. *IEEE Transactions on Software Engineering*, 39(5):658–683, DOI: 10.1109/TSE.2012.64.

Alexeeva, Z., Perez-Palacin, D., and Mirandola, R. (2016). Design decision documentation: A literature overview. *Software Architecture*, pages 84–101, ISBN: 978-3-319-48991-9, DOI: 10.1007/978-3-319-48992-6_6.

Anthony, A. (2016). NORDIC APIS - Tracking the Growth of the API Economy. `http://nordicapis.com/tracking-the-growth-of-the-api-economy`. Retrieved August 8, 2016.

Antonelli, M., Dellepiane, S. G., and Goccia, M. (2006). Design and implementation of web-based systems for image segmentation and cbir. *IEEE Transactions on Instrumentation and Measurement*, 55(6), DOI: 10.1109/TIM.2006.884286.

Anttonen, M., Salminen, A., Mikkonen, T., and Taivalsaari, A. (2011). Transforming the web into a real application platform: new technologies, emerging trends and missing pieces. In *proceedings of the 2011 ACM Symposium on Applied Computing (SAC '11)*, pages 800–807, Taichung, Taiwan. DOI: 10.1145/1982185.1982357.

Apache Software Foundation (2016a). Apache Ant. `http://ant.apache.org`. Retrieved May 24, 2016.

Apache Software Foundation (2016b). Apache Axis2/Java. `http://axis.apache.org/axis2/java/core/`. Retrieved May 24, 2016.

Apache Software Foundation (2016c). Apache Hadoop. `http://hadoop.apache.org`. Retrieved May 17, 2016.

Apache Software Foundation (2016d). Apache Solr. `http://lucene.apache.org/solr`. Retrieved May 17, 2016.

Apache Software Foundation (2016e). Apache Tomcat. `http://tomcat.apache.org`. Retrieved May 24, 2016.

Asana (2017). Asana work and project management. `https://asana.com/`. Retrieved January 17, 2017.

Atrey, P. K., Hossain, M. A., Saddik, A. E., and Kankanhalli, M. S. (2010). Multimodal fusion for multimedia analysis: a survey. *Multimedia Systems*, 16(6):345–379, ISSN: 0942-4962, DOI: 10.1007/s00530-010-0182-0.

Babar, M. A. and Gorton, I. (2004). A survey on software architecture evaluation methods. In *proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC '04)*, pages 600–607, Busan, Korea. DOI: 10.1109/APSEC.2004.38.

Bachmann, F., Bass, L., Carriere, J., Clements, P. C., Garlan, D., Ivers, J., Nord, R., and Little, R. (2000). Software architecture documentation in practice: Documenting architectural layers. technical note, Carnegie Mellon University.

Bachmann, F. and Merson, P. (2005). Experience using the web-based tool wiki for architecture documentation. technical note, Carnegie Mellon University.

Bass, L., Clements, P., and Kazman, R. (2012). *Software Architecture in Practice*. Addison-Wesley Professional, 3rd edition, ISBN: 978-0321815736.

Belwasmi, F., Glitho, R., and Chuyan, F. (2011). Restful web services for service provisioning in next-generation networks: A survey. *IEEE Communications Magazine*, 49(12):66–73, DOI: 10.1109/MCOM.2011.6094008.

Birrell, A. D. and Nelson, B. J. (1984). Implementing remote procedure calls. *ACM Transactions on Computer Systems (TOCS)*, 2(1):39–59, DOI: 10.1145/2080.357392.

Bosch, J. (2004). Software architecture: The next step. *Software Architecture*, pages 194–199, ISBN: 978-3-540-22000-8, DOI: 10.1007/978-3-540-24769-2_14.

Bosch, J. and Molin, P. (1999). Software architecture design: evaluation and transformation. In *proceedings of the IEEE Conference and Workshop on Engineering of Computer-Based Systems*, pages 4–10, Nashville, Tennessee, USA. DOI: 10.1109/ECBS.1999.755855.

Brambilla, M., Ceri, S., and Riccio, M. P. A. (2004). Managing asynchronous web services interactions. In *proceedings of the IEEE International Conference on Web Services*, pages 80–87, San Diego, California, USA. DOI: 10.1109/ICWS.2004.1314726.

Camera & Imaging Products Association (2015). Exchangeable image file format for digital still cameras: Exif Version 2.3, CIPA DC-008-Translation-2012, Standard of the Camera & Imaging Products Association. `http://www.cipa.jp/std/documents/e/DC-008-2012_E.pdf`. Retrieved September 20, 2015.

Cardinaels, K., Meire, M., and Duval, E. (2005). Automating metadata generation: the simple indexing interface. In *proceedings of the 14th international conference on World Wide Web*, Chiba, Japan. DOI: 10.1145/1060745.1060825.

Castillo, P. A., García-Sánchez, P., Arenas, M. G., Bernier, J. L., and Merelo, J. J. (2012). Distributed evolutionary computation using soap and rest web services. *Advances in Intelligent Modelling and Simulation*, pages 89–111, ISBN: 978-3-642-30153-7, DOI: 10.1007/978-3-642-30154-4_5.

Chen, L., Babar, M. A., and Nuseibeh, B. (2013). Characterizing architecturally significant requirements. *IEEE Software*, 30(2):38 – 45, ISSN: 0740-7459, DOI: 10.1109/MS.2012.174.

Cleland-Huang, J., University, D., Hanmer, R. S., Supakkul, S., and Mirakhorli, M. (2013). The twin peaks of requirements and architecture. *IEEE Software*, 30(2):24 – 29, ISSN: 0740-7459, DOI: 10.1109/MS.2013.39.

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R., and Stafford, J. (2010). *Documenting Software Architectures: Views and Beyond*. Addison-Wesley Professional, 2nd edition, ISBN: 978-0321552686.

Danielsen, P. J. and Jeffrey, A. (2013). Validation and interactivity of web api documentation. In *proceedings of the 2013 IEEE 20th International Conference on Web Services (ICWS '13)*, pages 523 – 530, Santa Clara, California, USA. DOI: 10.1109/ICWS.2013.76.

Datta, R., Joshi, D., Li, J., and Wang, J. Z. (2008). Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys*, 40(2), DOI: 10.1145/1348246.1348248.

Daughtry, J. M., Farooq, U., Myers, B. A., and Stylos, J. (2009). Api usability: Report on special interest group at chi. *ACM SIGSOFT Software Engineering Notes*, 34(4):27 – 29, DOI: 10.1145/1543405.1543429.

de Graaf, K. A., Tang, A., Liang, P., and van Vliet, H. (2012). Ontology-based software architecture documentation. In *proccedings of the 2012 Joint Working Conference on Software Architecture & 6th European Conference on Software Architecture (WICSA-ECSA '12)*, pages 121 – 130, Helsinki, Finland. DOI: 10.1109/WICSA-ECSA.212.20.

de Souza, C. R. B., Redmiles, D., Cheng, L.-T., Millen, D., and Patterson, J. (2004). Sometimes you need to see through walls - a field study of application programming interfaces. In *proceedings of the 2004 ACM conference on Computer supported cooperative work (CSCW '04)*, pages 63 – 71, Chicago, Illinois, USA. DOI: 10.1145/1031607.1031620.

de Souza, S. C. B., Anquetil, N., and de Oliveira, K. M. (2005). A study of the documentation essential to software maintenance. In *proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information (SIGDOC '05)*, pages 68 – 75, Coventry, United Kingdom. DOI: 10.1145/1085313.1085331.

Denning, P. J. (1997). A new social contract for research. *Communications of the ACM*, 40(2):132 – 134, DOI: 10.1145/253671.253755.

Digile (2015a). Data to Intelligence (D2I) SHOK program. `http://www.datatointelligence.fi`. Retrieved September 20, 2015.

Digile (2015b). FORGE Service Lab at GitHub. `https://github.com/forgeservicelab`. Retrieved September 20, 2015.

Dijkstra, E. W. (1982). *Selected Writings on Computing: A personal Perspective*. Springer New York, ISBN: 978-1-4612-5697-7, DOI: 10.1007/978-1-4612-5695-3.

Dilley, J., Maggs, B., Parikh, J., Prokop, H., Sitaraman, R., and Weihl, B. (2002). Globally distributed content delivery. *IEEE Internet Computing*, 50(58):50 – 58, DOI: 10.1109/MIC.2002.1036038.

Dimecc (2016). FORGE Service Lab, asset links. `http://www.dimecc.com/forge-service-lab-discontinued/`. Retrieved October 18, 2016.

Dobrica, L. and Niemelä, E. (2002). A survey on software architecture analysis methods. *IEEE Transactions on Software Engineering*, 28(7):638 – 653, ISSN: 0098-5589, DOI: 10.1109/TSE.2002.1019479.

Dubey, S. K. and Rana, A. (2011). Assessment of maintainability metrics for object-oriented software system. *ACM SIGSOFT Software Engineering Notes*, 36(5):1 – 7, ISSN: 0163-5948, DOI: 10.1145/2020976.2020983.

Ecma International (2013). Standard ECMA-404 – The JSON Data Interchange Format . `http://www.ecma-international.org/publications/standards/Ecma-404.htm`. Retrieved July 25, 2016.

Emery, D. and Hilliard, R. (2009). Every Architecture Description Needs a Framework: Expressing Architecture Frameworks Using ISO/IEC 42010. In *proccedings of the Joint Working IEEE/IFIP Conference on Software Architecture, 2009 & European Conference on Software Architecture (WICSA/ECSA 2009)*, pages 31 – 40, Cambridge, United Kingdom. DOI: 10.1109/WICSA.2009.5290789.

Espinha, T., Zaidman, A., and Gross, H.-G. (2014). Web api growing pains: Stories from client developers and their code. In *proceedings of the IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week*, pages 84 – 93, Antwerp, Belgium. DOI: 10.1109/CSMR-WCRE.2014.6747228.

Feng, X., Shen, J., and Fan, Y. (2009). Rest: An alternative to rpc for web services architecture. In *proceedings of the 1st International Conference on Future Information Networks (ICFIN 2009)*, pages 7 – 10, Beijing, China. DOI: 10.1109/ICFIN.2009.5339611.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, California, USA, ISBN: 0-599-87118-0.

Fielding, R. T. (2008). REST APIs must be hypertext-driven. `http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven`. Retrieved July 20, 2016.

Fielding, R. T. and Taylor, R. N. (2002). Principled design of the modern web architecture. *ACM Transactions on Internet Technology*, 2(2):115 – 150, DOI: 10.1145/514183.514185.

Forss, T., Liu, S., and Björk, K.-M. (2014). Extracting people's hobby and interest information from social media content. In *proceedings of Terminology and Knowledge Engineering 2014*, Berlin, Germany.

Fu, Z., Lu, G., Ting, K. M., and Zhang, D. (2011). A survey of audio-based music classification and annotation. *IEEE Transactions on Multimedia*, 13(2):303 – 319, ISSN: 1520-9210, DOI: 10.1109/TMM.2010.2098858.

Fuller, R. B. (1957). A comprehensive anticipatory design science. *Journal - Royal Architectural Institute of Canada*, 34, ISSN: 0383-6835.

Gerdes, S., Jasser, S., Riebisch, M., Schröder, S., Soliman, M., and Stehle, T. (2016). Towards the essentials of architecture documentation for avoiding architecture erosion. In *proccedings of the 10th European Conference on Software Architecture Workshops (ECSAW '16)*, Copenhagen, Denmark. DOI: 10.1145/2993412.3004844.

Gilliam, D. P., Wolfe, T. L., and Sherif, J. S. (2003). Software security checklist for the software life cycle. In *proceedings of the 12th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 243 – 248, Linz, Austria. DOI: 10.1109/ENABL.2003.1231415.

Glass, R. L. (2002). *Facts and Fallacies of Software Engineering*. Addison-Wesley Professional, 1st edition, ISBN: 978-0321117427.

Guinard, D., Ion, I., and Mayer, S. (2012). In search of an internet of things service architecture: Rest or ws-*? a developers' perspective. *Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pages 326 – 337, ISBN: 978-3-642-30972-4, DOI: 10.1007/978-3-642-30973-1_32.

Guo, L., Chen, S., Xiao, Z., and Zhang, X. (2005). Analysis of multimedia workloads with implications for internet streaming. In *proceedings of the 14th international conference on World Wide Web (WWW '05)*, pages 519 – 528, Chiba, Japan. DOI: 10.1145/1060745.1060821.

Gupta, V. and Lehal, G. S. (2009). A survey of text mining techniques and applications. *Journal of Emerging Technologies in Web Intelligence*, 1(1), ISSN: 1798-0461.

Hadar, I., Sherman, S., Hadar, E., and Harrison, J. J. (2013). Less is more: Architecture documentation for agile development. In *proccedings of the 2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, San Francisco, California, United States. DOI: 10.1109/CHASE.2013.6614746.

Hanbury, A. (2008). A survey of methods for image annotation. *Journal of Visual Languages and Computing*, 19(5):617 – 627, DOI: 10.1016/j.jvlc.2008.01.002.

Harrison, N. B. and Avgeriou, P. (2007). Leveraging architecture patterns to satisfy quality attributes. In *proceedings of the 1st European conference on Software Architecture (ECSA'07)*, pages 263 – 270, Madrid, Spain. ISBN: 978-3-540-75131-1.

Haslhofer, B. and Klas, W. (2010). A survey of techniques for achieving metadata interoperability. *ACM computing Surveys*, 42(2):617 – 627, DOI: 10.1145/1667062.1667064.

Henning, M. (2009). Api design matters. *Communications of the ACM - Security in the Browser*, 52(5):45 – 56, DOI: 10.1145/1506409.1506424.

Henningsson, K. and Wohlin, C. (2002). Understanding the relations between software quality attributes - a survey approach. In *proceedings of the 12th International Conference on Software Quality*, volume 12, pages 1 – 12, Ottawa, Canada.

Hevner, A. R. (2007). A three cycle view of design science research. *Scandinavian Journal of Information Systems*, 19(2):87 – 92.

Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). Design science in information systems research. *MIS Quarterly*, 28(1):75 – 105.

Hofmeister, C., Kruchten, P., Nord, R. L., Obbink, H., Ran, A., and America, P. (2005). Generalizing a model of software architecture design from five industrial approaches. In *proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, Pittsburgh, Pennsylvania, USA. DOI: 10.1109/WICSA.2005.36.

Hu, W., Xie, N., Li, L., Zeng, X., and Maybank, S. (2011). A survey on visual content-based video indexing and retrieval. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41(6):797 – 819, ISSN: 1094-6977, DOI: 10.1109/TSMCC.2011.2109710.

IETF (1976). Internet Engineering Task Force, RFC 707, A High-Level Framework for Network-Based Resource Sharing. `https://tools.ietf.org/html/rfc707`. Retrieved September 20, 2015.

IETF (1989). Internet Engineering Task Force - Network Working Group, RFC 1122, Requirements for Internet Hosts – Communication Layers. `https://tools.ietf.org/html/rfc1122`. Retrieved May 21, 2016.

IETF (2005). Internet Engineering Task Force - Network Working Group, RFC 4287, The Atom Syndication Format. `https://tools.ietf.org/html/rfc4287`. Retrieved May 24, 2016.

IETF (2014). Internet Engineering Task Force, RFC 7159, The JavaScript Object Notation (JSON) Data Interchange Format. `https://tools.ietf.org/html/rfc7159`. Retrieved July 25, 2016.

Immonen, A. and Niemelä, E. (2008). Survey of reliability and availability prediction methods from the viewpoint of software architecture. *Software & Systems Modeling*, 7(49), ISSN: 1619-1366, DOI: 10.1007/s10270-006-0040-x.

ISO/IEC (1994). International Organization for Standardization / International Electrotechnical Commission, ISO/IEC 7498-1:1994. Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model.

ISO/IEC (2002). International Organization for Standardization / International Electrotechnical Commission, ISO/IEC 15938-1:2002 Information technology – Multimedia content description interface – Part 1: Systems.

ISO/IEC (2004). International Organization for Standardization / International Electrotechnical Commission, ISO 8601:2004 Data elements and interchange formats – Information interchange – Representation of dates and times.

ISO/IEC (2011a). International Organization for Standardization / International Electrotechnical Commission, ISO/IEC 25010:2011. Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models.

ISO/IEC (2011b). International Organization for Standardization / International Electrotechnical Commission, ISO/IEC 42010:2011. Systems and software engineering - Architecture description.

ISO/IEC (2014). International Organization for Standardization / International Electrotechnical Commission, ISO/IEC 10646:2014 Information technology – Universal Coded Character Set (UCS).

Jansen, A., Avgeriou, P., and van der Ven, J. S. (2009). Enriching software architecture documentation. *Journal of Systems and Software*, 82(8):1232–1248, DOI: 10.1016/j.jss.2009.04.052.

JSON-RPC Working Group (2015). JSON-RPC 2.0 Specification. `http://www.jsonrpc.org/specification`. Retrieved September 20, 2015.

json.org (2016). JSON. `http://json.org/`. Retrieved July 25, 2016.

Junga, K., Kimb, K. I., and Jainc, A. K. (2004). Text information extraction in images and video: a survey. *Pattern Recognition*, 37(5):977–997, DOI: 10.1016/j.patcog.2003.10.012.

Kazman, R., Klein, M., and Clements, P. (2000). Atam: Method for architecture evaluation. technical report, Carnegie Mellon University.

Kitagawa, H., Ishikawa, Y., Li, Q., and Watanabe, C. (2010). Mediamatrix: A video stream retrieval system with mechanisms for mining contexts of query examples. *Database Systems for Advanced Applications*, pages 452–455, ISBN: 978-3-642-12097-8, DOI: 10.1007/978-3-642-12098-5_48.

Kiyoki, Y. and Chen, X. (2009). A semantic associative computation method for automatic decorative-multimedia creation with kansei information. In *proceedings of the Sixth Asia-Pacific Conference on Conceptual Modeling (APCCM '09)*, volume 96. Australian Computer Society, Inc., ISBN: 978-1-920682-77-4.

Kiyoki, Y., Kitagawa, T., and Hayama, T. (1994). A metadatabase system for semantic image search by a mathematical model of meaning. *ACM SIGMOD Record*, 23(4):34–41, DOI: 10.1145/190627.190639.

Kosanke, K. (2006). Iso standards for interoperability: a comparison. *Interoperability of Enterprise Software and Applications*, pages 55–64, ISBN: 978-1-84628-151-8, DOI: 10.1007/1-84628-152-0_6.

Kosch, H. and Maier, P. (2010). Content-based image retrieval systems - reviewing and benchmarking. *Journal of Digital Information Management*, 8(1):54–64.

Krippendorff, K. H. (2012). *Content Analysis: An Introduction to Its Methodology*. SAGE Publications, 3rd edition, ISBN: 978-1412983150.

Kruchten, P. (2009). Documentation of software architecture from a knowledge management perspective - design representation. *Software Architecture Knowledge Management - Theory and Practice*, pages 39–57, ISBN: 978-3-642-02373-6, DOI: 10.1007/978-3-642-02374-3_3.

Kruchten, P., Obbink, H., and Stafford, J. (2006). The past, present, and future of software architecture. *IEEE Software*, 23(2):22–30, ISSN: 0740-7459, DOI: 10.1109/MS.2006.59.

Lee, I. and Guan, L. (2004). Semi-automated relevance feedback for distributed content based image retrieval. In *proceedings of IEEE International Conference on Multimedia and Expo (ICME '04)*, pages 1871–1874, Taipei, Taiwan. DOI: 10.1109/ICME.2004.1394623.

Lew, M. S., Sebe, N., Djeraba, C., and Jain, R. (2006). Content-based multimedia information retrieval: State of the art and challenges. *ACM Transactions on Multimedia Computing, Communications and Applications*, 2(1):1–19, DOI: 10.1145/1126004.1126005.

Li, J., Xiong, Y., Liu, X., and Zhang, L. (2013). How does web service api evolution affect clients? In *proceedings of the 2013 IEEE 20th International Conference on Web Services (ICWS '13)*, pages 300–307, Santa Clara, California, USA. DOI: 10.1109/ICWS.2013.48.

Li, L. and Chou, W. (2011). Design and describe rest api without violating rest: A petri net based approach. In *proceedings of the 2011 IEEE International Conference on Web Services (ICWS)*, pages 508–515, Washington, D.C., USA. DOI: 10.1109/ICWS.2011.54.

Liang, J., Doermann, D., and Li, H. (2005). Camera-based analysis of text and documents: a survey. *International Journal of Document Analysis and Recognition (IJDAR)*, 7(2-3):84–104, ISSN: 1433-2833, DOI: 10.1007/s10032-004-0138-z.

Lourenço, J. R., Cabral, B., Carreiro, P., Vieira, M., and Bernardino, J. (2015). Choosing the right nosql database for the job: a quality attribute evaluation. *Journal of Big Data*, ISSN: 2196-1115, DOI: 10.1186/s40537-015-0025-0.

Lu, G. (2001). Indexing and retrieval of audio: A survey. *Multimedia Tools and Applications*, 15(3):269–290, ISSN: 1380-7501, DOI: 10.1023/A:1012491016871.

Ludwig, H., Laredo, J., Bhattacharya, K., Pasquale, L., and Wassermann, B. (2009). Rest-based management of loosely coupled services. In *proceedings of the 18th international conference on World wide web (WWW '09)*, pages 931–940, Madrid, Spain. DOI: 10.1145/1526709.1526834.

Lynx Technology (2016). Twonky. `http://www.lynxtechnology.com/twonky-overview`. Retrieved May 24, 2016.

Maleshkova, M., Pedrinaci, C., and Domingue, J. (2010). Investigating web apis on the world wide web. In *proceedings of the 2010 8th IEEE European Conference on Web Services (ECOWS '10)*, pages 107–114, Ayia Napa, Cyprus. DOI: 10.1109/ECOWS.2010.9.

Malik, A. and Nieminen, M. (2014). Understanding the usage and requirements of the photo tagging system. *Human IT*, 12(3):117–161, ISSN: 1402-150X.

Maramba, I. D., Davey, A., Elliott, M. N., Roberts, M., Roland, M., Brown, F., Burt, J., Boiko, O., and Campbell, J. (2015). Web-based textual analysis of free-text patient experience comments from a survey in primary care. *JMIR (Journal of Medical Internet Research) medical informatics*, 3(2), DOI: 10.2196/medinform.3783.

March, S. T. and Smith, G. F. (1995). Design and natural science research on information technology. *Decision Support Systems*, 15(4):251–266, DOI: 10.1016/0167-9236(94)00041-2.

Markus, M. L., Majchrzak, A., and Gasser, L. (2002). A design theory for systems that support emergent knowledge processes. *MIS Quarterly*, 26(3):179–212, ISSN: 0276-7783.

Matinlassi, M. (2006). *Quality-Driven Software Architecture Model Transformation. Towards Automation.* PhD thesis, VTT Technical Research Centre of Finland / University of Oulu, Oulu, Finland, ISBN: 951-38-6848-6.

May, N. (2005). A survey of software architecture viewpoint models. In *proccedings of the The Sixth Australasian Workshop on Software and System Architectures (AWSA 2005)*, pages 13 – 24, Brisbane, Australia.

McGraw, G. (2002). Managing software security risks. *Computer*, 35(4):99 – 101, DOI: 10.1109/MC.2002.993782.

McGraw, G. R. (2006). *Software Security: Building Security In.* Addison-Wesley Professional, 1st edition, ISBN: 978-0321356703.

Medvidovic, N. and Taylor, R. N. (2000). A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70 – 93, DOI: 10.1109/32.825767.

Microsoft Corporation (2003). *Improving Web Application Security: Threats and Countermeasures.* Microsoft Press, 1st edition, ISBN: 978-0735618428.

Microsoft Developer Network (2016). Chapter 16: Quality Attributes. `https://msdn.microsoft.com/en-us/library/ee658094.aspx`. Retrieved May 17, 2016.

Microsoft Technet (2016). TCP/IP Protocol Architecture. `https://technet.microsoft.com/en-us/library/cc958821.aspx`. Retrieved May 21, 2016.

Money, A. G. and Agius, H. (2008). Video summarisation: A conceptual framework and survey of the state of the art. *Journal of Visual Communication and Image Representation*, 19(2):121 – 143, DOI: 10.1016/j.jvcir.2007.04.002.

Mulligan, G. and Gračanin, D. (2009). A comparison of soap and rest implementations of a service based interaction independence middleware framework. In *proceedings of the 2009 Winter Simulation Conference (WSC)*, pages 1423 – 1432, Austin, Texas, USA. DOI: 10.1109/WSC.2009.5429290.

Multimedia, D. (2016). D2I presentation videos. `https://www.youtube.com/playlist?list=PLOHi43f5MZrjfzKHfujh9eO60EDk80Qb9`. Retrieved October 19, 2016.

MUVIS (2015). A System for Content-Based Indexing and Retrieval in Multimedia Databases. `http://muvis.cs.tut.fi`. Retrieved September 20, 2015.

Müller, H., Müller, W., Marchand-Maillet, S., Pun, T., and Squire, D. M. (2003). A framework for benchmarking in cbir. *Multimedia Tools and Applications*, 21:55 – 73, DOI: 10.1023/A:1025034215859.

Nagamachi, M. (2010). *Kansei/Affective Engineering.* CRC Press, ISBN: 978-1-4398-2133-6.

Navon, J. and Fernandez, F. (2011). The essence of rest architectural style. *REST: From Research to Practice*, pages 21 – 33, ISBN: 978-1-4419-8302-2, DOI: 10.1007/978-1-4419-8303-9_1.

Nord, R., Clements, P. C., Emery, D., and Hilliard, R. (2009). A structured approach for reviewing architecture documentation. technical report, Carnegie Mellon University.

Ntoulas, A., Najork, M., Manasse, M., and Fetterly, D. (2006). Detecting spam web pages through content analysis. In *proceedings of the 15th international conference on World Wide Web (WWW '06)*, pages 83 – 92, Edinburgh, Scotland, United Kingdom. DOI: 10.1145/1135777.1135794.

Oracle Corporation (2016a). MySQL Cluster CGE. `http://www.mysql.com/products/cluster/`. Retrieved May 17, 2016.

Oracle Corporation (2016b). MySQL Community Server. `http://dev.mysql.com/downloads/mysql/`. Retrieved May 24, 2016.

O'Brien, L., Merson, P., and Bass, L. (2007). Quality attributes for service-oriented architectures. In *proceedings of the International Workshop on Systems Development in SOA Environments (SDSOA '07)*, Minneapolis, Minnesota, USA. DOI: 10.1109/SDSOA.2007.10.

O'Reilly Media, Inc. (2016). xml.com. `http://www.xml.com/pub/rg/XML_Parsers`. Retrieved July 25, 2016.

Pallis, G. and Vakali, A. (2006). Insight and perspectives for content delivery networks. *Communications of the ACM - Personal information management*, 49(1):101 – 106, DOI: 10.1145/1107458.1107462.

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053 – 1058, DOI: 10.1145/361598.361623.

Patidar, A. and Suman, U. (2015). A survey on software architecture evaluation methods. In *proceedings of the 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 967 – 972, New Delhi, India. ISBN: 978-9-3805-4415-1.

Paulson, L. D. (2005). Building rich web applications with ajax. *Computer*, 38(10):14 – 17, DOI: 10.1109/MC.2005.330.

Pautasso, C. (2009). Restful web service composition with bpel for rest. *Data & Knowledge Engineering*, 68(9):851 – 866, DOI: 10.1016/j.datak.2009.02.016.

Pautasso, C., Zimmermann, O., and Leymann, F. (2008). Restful web services vs. 'big' web services: making the right architectural decision. In *proceedings of the 17th International Conference on World Wide Web*, Beijing, China. DOI: 10.1145/1367497.1367606.

Peffers, K., Tuunanen, T., Rothenberger, M., and Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, 24(3):45 – 77, ISSN: 1402-150X, DOI: 10.2753/MIS0742-1222240302.

Perry, D. E. and Wolf, A. L. (1992). Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40 – 52, ISSN: 0163-5948, DOI: 10.1145/141874.141884.

Pivotal Software (2016). Spring Framework. `https://projects.spring.io/spring-framework`. Retrieved May 24, 2016.

Rahman, M. M., Bhattacharya, P., and Desai, B. C. D. (2007). A framework for medical image retrieval using machine learning and statistical similarity matching techniques with relevance feedback. *IEEE Transactions on Information Technology in Biomedicine*, 11(1), DOI: 10.1109/TITB.2006.884364.

Rantanen, P. and Sillberg, P. (2009). Mobiilia päätelaitetta hyödyntävä viestinvälitysratkaisu katastrofi- ja hätätilanteisiin. Master's thesis, Tampere University of Technology.

Rantanen, P., Sillberg, P., Soini, J., and Jaakkola, H. (2017). Tag suggestions from social media profiles. *Information Modelling and Knowledge Bases XXVIII*, pages 354 – 361, ISBN: 978-1-61499-719-1, DOI: 10.3233/978-1-61499-720-7-354.

Reade, C. (1989). *Elements Of Functional Programming*. Addison-Wesley, 1st edition, ISBN: 978-0201129151.

Red Hat, Inc (2016). Ansible is Simple IT Automation. `https://www.ansible.com`. Retrieved May 24, 2016.

Richardson, L. and Ruby, S. (2007). *RESTful web service*. O'Reilly Media, ISBN: 978-0-596-52926-0.

Riva, C. and Laitkorpi, M. (2009). Designing web-based mobile services with rest. *Service-Oriented Computing - ICSOC 2007 Workshops*, pages 439 – 450, ISBN: 978-3-540-93850-7, DOI: 10.1007/978-3-540-93851-4_42.

Robles, O. D., Bosque, J. L., Pastor, L., and Rodriguez, A. (2005). Performance analysis of a cbir system on shared-memory systems and heterogeneous clusters. In *proceedings of the Seventh International Workshop on Computer Architecture for Machine Perception (CAMP'05)*, Palermo, Italy. DOI: 10.1109/CAMP.2005.40.

Rost, D., Naab, M., Lima, C., and von Flach Garcia Chavez, C. (2013). Software architecture documentation for developers: A survey. In *proccedings of the 7th European conference on Software Architecture (ECSA'13)*, pages 72 – 88, Montpellier, France. DOI: 10.1007/978-3-642-39031-9_7.

Roy, B. and Graham, T. N. (2008). Methods for evaluating software architecture: A survey. technical report, School of Computing, Queen's University at Kingston, Ontario, Canada.

Salton, G. (1968). Automatic content analysis in information retrieval. technical report, Cornell University.

Schedl, M., Gómez, E., and Urbano, J. (2014). Music information retrieval: Recent developments and applications. *Foundations and Trends in Information Retrieval*, 8(2-3):127 – 261, ISBN: 978-1-60198-807-2.

Sillberg, P., Rantanen, P., and Soini, J. (2013). A content based tool for searching, connecting and combining digital information - case: Smart photo service. In *proceedings of the 16th International Multiconference Information Society (IS 2013)*, volume A, pages 249 – 252, Ljubljana, Slovenia. Josef Stefan Institute.

Simon, H. A. (1996). *The Sciences of the Artificial*. The MIT Press, 3rd edition, ISBN: 978-0262691918.

Smith, K. (2006). Simplifying ajax-style web development. *Computer*, 39(5):98 – 101, DOI: 10.1109/MC.2006.177.

Soini, J., Sillberg, P., and Rantanen, P. (2011). Deployment of new techniques for searching, sorting, processing and presenting data. In *proceedings of the 14th International Multiconference Information Society (IS 2011)*, volume A, pages 179 – 182, Ljubljana, Slovenia. International Multiconference Information Society IS.

Swanberg, D., Shu, C. F., and Jain, R. (1992). Architecture of a multimedia information system for content-based retrieval. In *proceedings of the Third International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 387 – 392, La Jolla, California, USA.

Tang, A., Babar, M. A., Gorton, I., and Han, J. (2006). A survey of architecture design rationale. *Journal of Systems and Software*, 79(12):1792 – 1804, DOI: 10.1016/j.jss.2006.04.029.

Tarvainen, P. (2007). Adaptability evaluation of software architectures; a case study. In *proceedings of the 31st Annual International Computer Software and Applications Conference*, volume 2, pages 579 – 586, Beijing, China. DOI: 10.1109/COMPSAC.2007.240.

Taylor, R. N., Medvidovic, N., and Dashofy, E. M. (2009). *Software Architecture: Foundations, Theory, and Practice*. John Wiley & Sons, ISBN: 978-0470167748.

The Bug Genie (2017). The bug genie - friendly issue tracking and project management. `http://thebuggenie.com/`. Retrieved January 17, 2017.

The jQuery Foundation (2016). jquery. `https://jquery.com/`. Retrieved May 18, 2016.

Tipaldo, G. (2014). *L'analisi del contenuto e i mass media*. Il Mulino, ISBN: 978-8815248329.

Trojacanec, K., Dimitrovski, I., and Loskovska, S. (2009). Content based image retrieval in medical applications: An improvement of the two-level architecture. In *proceedings of IEEE Eurocon 2009*, pages 118 – 121, St. Petersburg, Russia. DOI: 10.1109/EURCON.2009.5167614.

Tsichritzis, D. (1997). The dynamics of innovation. *The Next Fifty Years of Computing*, pages 259 – 265, ISBN: 978-0-387-98588-6, DOI: 10.1007/978-1-4612-0685-9_19.

Tsytsarau, M. and Palpanas, T. (2011). Survey on mining subjective data on the web. *Data Mining and Knowledge Discovery*, 24(3):478 – 514, ISSN: 1384-5810, DOI: 10.1007/s10618-011-0238-6.

TUT Pori (2015a). Tampere University of Technology, Pori Department. CAFrontEnd Specification. `http://visuallabel.github.io/javadoc/visuallabel/documents.html`. Retrieved January 17, 2017.

TUT Pori (2015b). Tampere University of Technology, Pori Department. VisualLabel at GitHub. `https://github.com/visuallabel`. Retrieved September 20, 2015.

TUT Pori (2015c). Tampere University of Technology, Pori Department. VisualLabel at GitHub, Analysis Back End Capabilities. `http://visuallabel.github.io/javadoc/visuallabel/service/tut/pori/contentanalysis/AnalysisBackend.Capability.html`. Retrieved May 16, 2016.

TUT Pori (2015d). Tampere University of Technology, Pori Department. VisualLabel at GitHub, Content Analysis Use Cases. `https://github.com/visuallabel/CAFrontEnd/wiki/Use-Cases`. Retrieved May 18, 2016.

TUT Pori (2015e). Tampere University of Technology, Pori Department. VisualLabel at GitHub, Data Groups. `http://visuallabel.github.io/javadoc/visuallabel/core/tut/pori/http/parameters/DataGroups.html`. Retrieved May 16, 2016.

TUT Pori (2015f). Tampere University of Technology, Pori Department. VisualLabel at GitHub, Photo Task Example. `http://visuallabel.github.io/javadoc/visuallabel/service/tut/pori/contentanalysis/PhotoTaskDetails.html`. Retrieved May 16, 2016.

TUT Pori (2015g). Tampere University of Technology, Pori Department. VisualLabel at GitHub, wikipages. `https://github.com/visuallabel/CAFrontEnd/wiki/What-is-VisualLabel%3F`. Retrieved January 17, 2017.

TUT Pori (2016a). Tampere University of Technology, Pori Department. Javadocer source codes for VisualLabel doclet extensions. `https://github.com/visuallabel/javadocer`. Retrieved January 17, 2017.

TUT Pori (2016b). Tampere University of Technology, Pori Department. VisualLabel at GitHub, A simple web page for testing the basic HTTP operations. `https://github.com/visuallabel/CAFrontEnd/blob/master/web/debug/poster.jsp`. Retrieved May 18, 2016.

TUT Pori (2016c). Tampere University of Technology, Pori Department. VisualLabel at GitHub, Deployment Guide. `https://github.com/visuallabel/CAFrontEnd/wiki/Deployment-Guide`. Retrieved May 18, 2016.

Typke, R., Wiering, F., and Veltkamp, R. C. (2005). A survey of music information retrieval systems. In *proceedings of the 6th International Conference on Music Information Retrievals (ISMIR 2006)*, London, United Kingdom.

Unicode, Inc. (2016). The Unicode Standard. `http://www.unicode.org/standard/standard.html`. Retrieved July 25, 2016.

Valipour, M. H., Amirzafari, B., Maleki, K. N., and Daneshpour, N. (2009). A brief survey of software architecture concepts and service oriented architecture. In *proceedings of the 2nd IEEE International Conference on Computer Science and Information Technology, (ICCSIT 2009)*, pages 34 – 38, Beijing, China. DOI: 10.1109/ICCSIT.2009.5235004.

van Heesch, U., Avgeriou, P., and Hilliard, R. (2012). A documentation framework for architecture decisions. *Journal of Systems and Software*, 85(4):795 – 820, DOI: 10.1016/j.jss.2011.10.017.

Verborgh, R., Harth, A., Maleshkova, M., Stadtmüller, S., Steiner, T., Taheriyan, M., and de Walle, R. V. (2014). Survey of semantic description of rest apis. *REST: Advanced Research Topics and Practical Applications*, pages 69 – 89, ISBN: 978-1-4614-9298-6, DOI: 10.1007/978-1-4614-9299-3_5.

Vogel, J. and Schiele, B. (2006). Performance evaluation and optimization for content-based image retrieval. *Pattern Recognition*, 39(5):897 – 909, DOI: 10.1016/j.patcog.2005.10.024.

W3C (2004a). World Wide Web Consortium, Web Services Architecture. http://www.w3.org/TR/ws-arch/. Retrieved July 19, 2016.

W3C (2004b). World Wide Web Consortium, Web Services Glossary, W3C Working Group Note. http://www.w3.org/TR/ws-gloss. Retrieved September 20, 2015.

W3C (2007a). World Wide Web Consortium, SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), W3C Recommendation. http://www.w3.org/TR/soap12-part1/. Retrieved September 20, 2015.

W3C (2007b). World Wide Web Consortium, Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, W3C Recommendation. http://www.w3.org/TR/wsdl20. Retrieved September 20, 2015.

W3C (2008). World Wide Web Consortium, Extensible Markup Language (XML) 1.0 (Fifth Edition). https://www.w3.org/TR/xml/. Retrieved July 25, 2016.

Walls, J. G., Widmeyer, G. R., and Sawy, O. A. E. (1992). Building an information system design theory for vigilant eis. *Information Systems Research*, 3(1):36 – 59, ISSN: 1526-5536, DOI: 10.1287/isre.3.1.36.

Wang, H. and Wang, C. (2003). Taxonomy of security considerations and software quality. *Communications of the ACM - E-services: a cornucopia of digital offerings ushers in the next Net-based evolution*, 46(6), DOI: 10.1145/777313.777315.

Web Sequence Diagrams (2017). Web sequence diagrams. https://www.websequencediagrams.com/. Retrieved January 17, 2017.

Wikipedia (2016). Content analysis. https://en.wikipedia.org/wiki/Content_analysis. Retrieved April 28, 2016.

Yoshitaka, A. and Ichikawa, T. (1999). A survey on content-based retrieval for multimedia databases. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):81 – 93, ISSN: 1041-4347, DOI: 10.1109/69.755617.

Zdun, U., Völter, M., and Kircher, M. (2004). Pattern-based design of an asynchronous invocation framework for web services. *International Journal of Web Services Research*, 1(3):42 – 62, DOI: 10.4018/jwsr.2004070103.

Zettsu, K. (2012). Knowledge processing technology: An overview of knowledge cluster systems. *Journal of the National Institute of Information and Communications Technology*, 59(3/4):155 – 168.

Zhang, D., Islam, M. M., and Lu, G. (2012). A review on automatic image annotation techniques. *Pattern Recognition*, 45(1):346 – 362, DOI: 10.1016/j.patcog.2011.05.013.

Publications

Publication I

Sillberg, P., **Rantanen P.**, Saari, M., Leppäniemi, J., Soini, J. and Jaakkola, H., "Towards an IP-Based Alert Message Delivery System", in Proceedings of the 6th International ISCRAM Conference, ISBN 978-91-633-4715-3, Gothenburg, Sweden, May 10-13, 2009.

Towards an IP-Based Alert Message Delivery System

Pekka Sillberg
Tampere University of Technology (TUT), Pori
Finland
pekka.sillberg@tut.fi

Petri Rantanen
Tampere University of Technology (TUT), Pori
Finland
petri.rantanen@tut.fi

Mika Saari
Tampere University of Technology (TUT), Pori
Finland
mika.saari@tut.fi

Jari Leppäniemi
Tampere University of Technology (TUT), Pori
Finland
jari.leppaniemi@tut.fi

Jari Soini
Tampere University of Technology (TUT), Pori
Finland
jari.o.soini@tut.fi

Hannu Jaakkola
Tampere University of Technology (TUT), Pori
Finland
hannu.jaakkola@tut.fi

ABSTRACT

Advancements in technology have provided new opportunities for the delivery of emergency messages. However, some of the issues concerning data security and technical solutions are quite different from the problems of the traditional means of communication. The Internet poses its own set of challenges. This paper presents a few emergency messaging system proposals made by other researchers and also introduces a new proposition put forward by the authors of this paper. This will demonstrate how to use client-server architecture to deliver emergency alert messages in IP-based networks. The proposed system uses Atom feeds to deliver alert messages and also provides a feedback channel for client data. In this scenario clients could have any kind of device from mobile terminals to desktop computers.

Keywords

IP-based alert message, Mobile Emergency Announcement, Common Alerting Protocol, Atom feed, Information systems

INTRODUCTION

Mobile terminals have become an essential part of everyday life. The penetration level of mobile phones is approaching 100% in well-developed economies and even in less developed economies the number of phones is rising very quickly. From this aspect, the use of mobile phones for delivering emergency information is justified. In this paper the objective is how to utilize mobile technology to supply disaster information and especially communication by mobile phone. We have worked with a communication system that delivers alert messages. One main research area is how to send alert messages to mobile devices. This paper describes the communication mechanism which is needed to send and receive alert messages in mobile devices.

The major challenges of emergency management are to reduce the human and economic costs of the emergency. One of the most important elements to achieve this goal is to make the general public aware of and prepare them for an upcoming emergency situation. According to the Organization for the Advancement of Structured Information Standards (OASIS, 2005), the emergency management cycle consists of the four main phases shown in Figure 1.

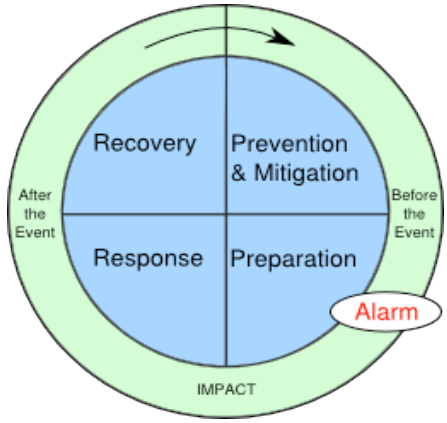

Figure 1. The Disaster and Emergency cycle (Adapted from OASIS, 2005; Alexander, 2002)

The purpose of the Prevention and Mitigation phase is to lower the possibility of the disaster – or emergency – occurring and/or to minimize the damage caused. Actions done for this phase are generally long-term processes. The Preparation phase typically consists of actions taken when a disaster is anticipated or impending in order to ensure a rapid and more effective response. Actions taken during the Response phase are aimed at saving human life, the protection of assets, the supply of vital goods and services, and the protection of the environment. The Response phase begins as the disaster/emergency strikes and continues until the Recovery phase takes over. Recovery is the process by which communities return to a normal level of functioning. (OASIS, 2005)

One of the most important features of an emergency alert system – regardless of what medium/media it uses as its transfer layer – should be its ability to deliver the emergency alert effectively to as many of the people under threat as possible. According to a study made by Verma and Verma (2005), a good emergency alert system should have the properties of locality, automated operation, non-intrusiveness, spontaneity, ubiquity and support for second languages. These properties are not tied only to emergency alert systems working over the Internet, thus they may be applied to any emergency alert system. Locality means that the emergency alert should be available to the general population that is affected by the emergency. The automated operation of the emergency alert system means that it should be able to switch to alert mode from normal mode and vice versa without the need for manual intervention. Non-intrusiveness and spontaneity are especially aimed at emergency alert systems but may also be applied to "normal" emergency alert systems. Non-intrusiveness means that the user's activities should not be disrupted by the alert system. When the alert system is spontaneous, it can deliver and show alerts to the user without the user needing to do any predetermined action manually. Ubiquity by definition means "being everywhere," thus in an alert system context it means that alerts should be provided to everyone affected, and should not miss out any user in the affected area. The support for a second language simply means that the emergency alert system should be able to provide alerts in another or many other languages.

SSMC/DDKM - The Project

The starting point of this study is the ongoing SSMC/DDKM (Seamless Services and Mobile Connectivity in Distributed Disaster Knowledge Management) research project, coordinated by the Tampere University of Technology (TUT). The general goal of this two-year research project (Soini, Leppäniemi and Jaakkola, 2008) is to study and develop the methods, processes and technologies to support improved knowledge management in disasters and accidents. The ability to build a reliable situational model based on the history and current knowledge of the situation and on good practices concerning comparable accidents/disasters is important in minimizing the consequences of an accident. The project is funded 70% by Tekes (the Finnish Funding Agency for Technology and Innovation) and 30% by a consortium consisting of two Finnish ICT companies and the Finnish Emergency Response Centre Agency. The SSMC/DDKM project includes two dimensions: *international* and *national.* The project is based on *international research* co-operation between several organizations. The leading forces are NICT (National Research Institute on Information and Communication Technology) and Keio University (SFC) in Japan.

The Japanese project has been given government financing for a 5-year period. The goal of this international part is to develop a distributed disaster knowledge management system, which supports the connectivity of separate knowledge sources (NICT, 2007a, 2007b). In the *national dimension* the project focuses on investigating current collaboration – taking into account both technical and social aspects – in accident and disaster situations between the Finnish authorities who are responsible for producing, disseminating and utilizing information in these situations.

The structure of this paper is the following: first, the background of the subject is introduced in the section entitled "Emergency alert systems on the Internet." The standards used are also mentioned. In the next section "Towards an IP-based alert Message Delivery System," the basic construction of our proposed emergency message alert system is described. At the beginning we describe the protocol used in client - server communication. The section entitled "Communication flow between server and client" includes two cases for messaging between client and server: a basic case and an extended case. The basic case is demonstrated by a sequence diagram of message transportation. The extended case demonstrates the use of a feedback channel. Finally the study is summarized in the "Conclusion".

EMERGENCY ALERT SYSTEMS ON THE INTERNET

What makes IP-based (such as Internet) emergency alert systems plausible are the many possibilities the Internet has to offer. For example more and more of the mobile devices sold nowadays have the option of connecting to the Internet using wireless networks – or other high bandwidth networks – and can also run programs made to support emergency messaging. This can enable services where one could report immediately from the disaster site thus giving first hand knowledge for use in emergency management. The service provided could easily be expanded to giving alerts, warnings or instructions before and after the emergency incident. IP-based alerting systems also allow other devices accessing the network to use the service, such as desktop computers and personal digital assistants (PDA). This is a great advantage over emergency alert systems designed solely for mobile phones (for example SMS messages).

When sending or receiving information over the Internet or any other IP-based network architecture, there are basically two possible implementations: The first one is to make a custom – free or proprietary – protocol and use it over the existing network infrastructure – and the second option is to use a standardized protocol. In many cases the first kind of implementation (proprietary protocol) may be the faster way of transmitting information and more optimized regarding bandwidth usage. However, the custom structure of the chosen protocol could cause problems when communicating with third party software or when interfacing with other systems. Therefore, especially with emergency messaging, this is not a desirable outcome. The interoperability with multiple systems, devices and software should be one of the prime goals of development.

To overcome the interoperability problems in emergency information transfer the second option is preferable, i.e. a commonly known and standardized protocol. For emergency messages the Common Alerting Protocol (CAP) of the Organization for the Advancement of Structured Information Standards (OASIS, 2008) is a good choice because of its standardized format (CAP v1.1, 2005; Botterell, 2006) and completely free and open nature. CAP has already been used in many official systems and especially in Atom feeds (CAP Cookbook, 2008; RFC 4287, 2005). CAP itself is a part of a larger protocol family known as Emergency Data Exchange Language (EDXL). EDXL has standardized means, for example, for sending messages using eXtensible Markup Language (XML) with and without the XML-based SOAP protocol (EDXL-DE v1.0, 2006). In conclusion, using standard means and protocols is the second and recommended implementation option. Both the Atom feed-based approach and the SOAP protocol are used in the emergency message system proposition presented in this paper. Next we will examine a few other systems that have also been developed for delivering emergency alerts.

There have been many previous propositions that use CAP or Atom or both for sending and receiving emergency messages. X-Capatom (IBM developerWorks, 2007) is quite similar to our proposal and there are also similarities with Art Botterell's Advanced EAS Relay Network (AERN) proposition's "Secure server" approach (Botterell, 2003). Botterell (2003) describes many other means for emergency message delivery and primarily these are meant to improve the United States' existing Emergency Message System (EAS), but the same ideas can be applied to other systems. When referring to AERN in this paper, the "Secure server" approach is meant. The AERN architecture is free and open for use for anyone, as is X-Capatom.

Both X-Capatom and AERN represent a system which has one or multiple secure servers and the content of these servers is updated by some authorized party. Since only selected individuals or parties can add or modify content on the server, the information can be solely trusted as long as the server is not compromised. This is the basic idea of the approach. X-Capatom does not especially mention the "Secure server" approach by name, but it can be

considered to belong to this category. AERN describes the architecture more and does not go too deeply into technical, code-level details.

X-Capatom in turn concentrates more on describing the basic functions and technical details of the system. IBM's X-Capatom pages (IBM developerWorks, 2007) even have a Java™ based reference implementation. As the name implies, X-Capatom uses a combination of Atom feeds and CAP to deliver messages. This approach has the major advantage of being able to deliver messages to many programs which are not originally meant to read emergency message information. Atom feeds can be read using many of the basic Really Simple Syndication (RSS) readers and by normal web browsers. It is also possible to make customized third party software that interprets the feeds in some specific and predefined way and displays the information to end users, but this is optional. X-Capatom uses Atom Publishing Protocol (RFC 5023, 2007) for modifying, deleting and adding new information to Atom feeds. The reference implementation does not have any kind of user authorization or user control even though the X-Capatom recommends adding such features when using the system. The feeds themselves are available for everyone to read.

In summary, the techniques and methods presented earlier in this section share many common elements and basic ideas with the emergency messaging system proposed by the authors of this paper. In next section the new proposal and implementation example are described in more detail.

TOWARDS AN IP-BASED ALERT MESSAGE DELIVERY SYSTEM

The system that we are introducing in this paper is fundamentally based on the integration of two different messaging channels working together in a single system. The diagram (Figure 2) below illustrates the basic operation of the system. In our example the *feed channel* uses Atom feeds as a transfer medium. The feeds are compiled by the server using data it has collected from various sources. These information sources could be basically any type. These feeds contain the elements of Atom and CAP – or any other XML-based format – elements with the necessary processing information for use by the client software. Feeds can be used as standalone feeds in any RSS/Atom reader and can still offer sufficient information about the incident for the user. In cases where the user does have an enhanced reader program, the processing information in the feed can offer extended features. For example, one feature could be an automatically initiated alert if a predefined threshold value is exceeded. Another example could be opening a map program showing the location of the coordinates included in the feed.

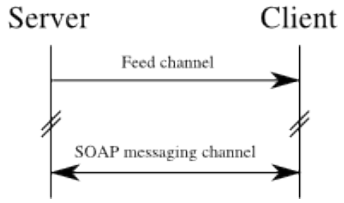

Figure 2. Simple communication sequence

The Atom feeds, i.e. the alert messages, should be constructed in such a way that every reader (normal RSS/Atom readers or enhanced clients) is capable of understanding all the Atom-specific elements of the feeds and seeing the crucial alert information. This means that processing information meant for enhanced clients should only include meta-information of the information already provided in the common part of the Atom feed. In this way different kinds of clients/devices receive the same information, and only the layout and the presentation vary. In case of the previously mentioned example of opening the map program, if the client device cannot show maps, the user can still see the coordinates and verbal descriptions of the location. This kind of difference in the visual presentation of the information may affect the intelligibility of the alert messages (e.g. it might be difficult for the user to figure out the precise geographical location of the coordinates).

In our example the extended messaging channel is using SOAP encapsulated messages. This channel allows users to initiate the data transfer between the server and client as well as the other way around. In other words, use of this kind of approach enables a feedback channel as well as a wide range of services and features. But this approach is

not completely free of problems and/or drawbacks. For instance, the server needs to know where its users are (IP addresses etc.) and some network applications such as Network Address Translators (NAT) and firewalls may interfere with connection attempts made by the server. Also, client devices running software for listening to a server, in order to make a connection, may make them vulnerable to security threats.

The emergency message delivery system proposal presented in this paper follows the ”Secure server” approach and the ”Client software” approach (Verma and Verma, 2005). The feed-based system was chosen because it enables normal users, that is, users who are not known to the system in any way to read emergency messages using whatever software they prefer. Atom feeds were chosen purely because of Atom's status as the Internet Engineering Task Force (IETF) standard (RFC 4287, 2005). RSS feeds may be used more often, but there is no common agreement about what RSS feeds can or cannot include. In some cases this kind of freedom can be a good thing, but when a standardized format is required it is not. Additionally, unlike Atom, RSS does not state what to do with XML elements and this is left to the XML processing programs to decide. Atom especially defines that unknown elements, fields or attributes should *not* be processed in anyway. This means that any software working as a client cannot start guessing what to do with this kind of unknown data. In emergency messaging it is crucial that unknown messages are not interpreted wrongfully or “guessed.” All client – and of course server-side – programs should process the messages similarly and notify the user in the case of unclear messages. This can be done with RSS, but it is *not required* by any standard. It should be noted that nothing guarantees that programs using Atom do not just ignore the unknown data and continue normally, but in well-made programs this kind of behavior is not desirable. Another gain in using Atom compared to RSS is its ability to clearly state whether the feed content is just a collection of elements from some other message (for example a CAP message) or the complete original message in the form of a link pointing to another destination (Westfall, 2007).

System overview

The picture below (Figure 3) shows a generalized overview of our proposal i.e. the system described in this paper. In essence, the purpose of the system is to relay messages. The system is used to relay emergency alert information, but the same basic model could be used with any kind of information

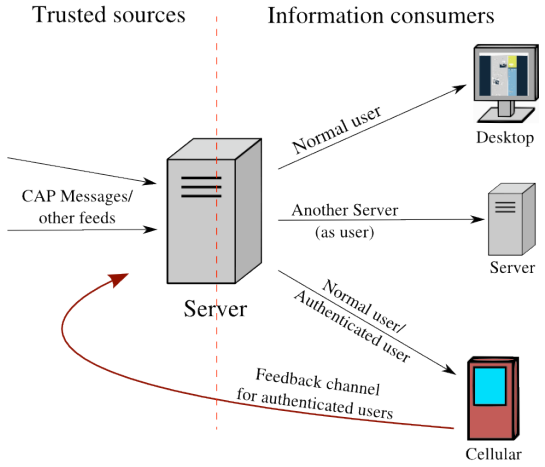

Figure 3. Overview of system

The “Server” is the connecting part between two sides, the trusted sources – which could be thought of as the providers of the information – and the information consumers. The major difference between this model and the X-Capatom is that in the basic usage scenario there is no need for user management or authorization. This is because no information is needed about the “normal users” reading the feeds – this is also true with X-Capatom. The providers of the information should also be trusted automatically. These trusted sources could be anything from CAP messages offered by official sources or other Atom feeds – RSS feeds are also possible despite their

nonstandard nature. This allows some servers to work as information providers for other servers, or in other words, they could behave like a client of a server.

All information coming from trusted sources should be transmitted over secure communication lines to guarantee that the messages are authentic. The secure communication could be anything from Virtual Private Networks (VPN) to Secure Sockets Layer / Transport Layer Security (SSL/TLS) using certificates (for example X.509 certificates). It is also possible to use certificates and unencrypted data when only message integrity is required. SSL/TLS is also a good choice for message authentication checks in Atom feeds because it requires very little changes in the operation of the web server, though it should be noted that using encryption causes extra workload on the server side. Unencrypted data and X.509 certificates are often enough in feeds because the information is meant to be read by anyone and there is no risk of information leaks to unwanted parties or individuals. If the system is used to relay confidential information that often means that the information is meant for a relatively small group of individuals and in that case the extra workload caused by data encryption is smaller.

The figure 3 (above) also shows the possibility of an authenticated user. Because the server can receive messages from the trusted sources side over an SSL/TLS connection, the same functionality enables it to receive messages from "Normal users." These users must be known to the server and authenticated before any information can be received. Connection to authenticated users can be thought of as a secondary means of information retrieval and can be used for example to transmit real-time, on-location data between different authorities (for example between the police and fire departments). Knowing certain clients and collecting information about them – such as IP addresses and/or geographic locations – also makes it possible to connect to these clients directly. In this paper's emergency message system the direct connection to clients and the feed-based information relay are separated to different usage cases and these cases are described below (Figure 4) in more detail. Direct connection includes both sending information to clients and receiving information from them.

Communication flow between server and client

The differences to existing systems are pointed out using a couple of example cases (*basic* and *extended)*. In principle most World Wide Web (WWW) pages (such as Web portals and news pages) use the basic approach where the client is solely responsible for fetching (i.e. pulling) the information. This is also how many other messaging services work. Extended cases use the push approach for transmitting information from sender to receiver, that is from client to server or from server to client.

Communication between the server and the client is quite simple and straightforward. The diagram below (Figure 4) describes the most common use cases offered by the system as a whole.

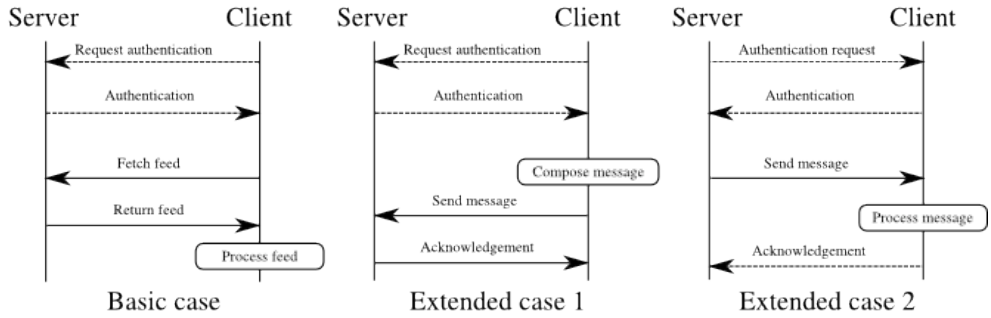

Figure 4. Sequence diagrams of most common use cases

In every case there is an optional authentication phase. During this phase both server and client may be authenticated and they may form a secure connection if necessary. In cases where the client makes the initiating connection to the server using an SSL or TLS connection, the server side can be verified by the certificate it uses. When the server initiates the connection, both sides may need to verify each other. For example, the challenge-response-authentication method may be used. Phases or actions with dashed lines can be skipped if they are not needed or if it

has been decided not to use them. Even though the authentication step can be skipped, it may not be advisable due to possible security threats.

Basic case

This is the most basic step (Figure 4 above, leftmost sequence diagram) that can be done in any reader that supports Atom feeds. Retrieval of the feed is done either by giving the direct access to a requested feed or by searching for a suitable feed from the available feeds. Firstly, the authentication step is made as described in the previous section. Then the feed will be retrieved to the client device. From that point the software decides how the feed is displayed. If the client software is a basic RSS reader, it will show the Atom content of the feed as it should. When a user is reading the feed with software designed to interpret the enhanced feed, not only will it show the content of the message(s), but also perform special features based on the extra information included in the feed.

Extended cases

These two cases (Figure 4 above, middle and rightmost sequence diagram) illustrate the use of the extended messaging channel. As a reminder, users of this messaging channel must be registered and authenticated. Therefore authenticated users are referred to as "users" below unless otherwise stated. In case number one (Figure 4, center sequence diagram), the client software initiates the connection and in the second case (Figure 4, rightmost sequence diagram) the server is the initiating side. In both cases a message, a reply to some earlier message, a command etc. can be delivered to the receiver. An example of using case 2 could be the following: the server receives a message that is a very important alert and certain user(s) must get it immediately. As a result the server opens the messaging channel to these clients and sends the alert to them. An example of case 1: the client software notices that the IP address bound to the device has changed for some reason. The client software then opens the messaging channel to the server and updates the user information to match the situation (this is just a simplified example and does not describe the steps that may need user intervention such as asking permission to open the data connection).

As both messaging channels are integrated in one system, they may seamlessly use information from/about the user, available feeds and other information to enable various emergency alerting services. Although the focus has been on emergency alerts and the like, the system is not limited to them alone. With proper extensions the system may be used for the delivery of traffic information, weather information and so on. Basically, the emergency alerting system could be just one small part of a larger information delivery system.

We believe that there are two promising features in the system introduced here. The first is the combination of two communication channels that allows a "simple" messaging service using one of the channels and the other channel enables the feedback from user to server. This also allows for additional possibilities such as executing certain features on the client device. The second feature is that the new system may be run virtually on any platform or system that is able to connect to the Internet. The next step we are taking in the research will be the implementation of the system as a proof-of-concept styled demo system. The demo is anticipated to be ready in summer 2009.

CONCLUSION

Mobile terminals have become an essential part of everyday life. These have provided new opportunities for the delivery of emergency messages using mobile phones. In this paper we have shown how to utilize mobile technology to supply disaster information to both mobile terminals and desktop computers. We have also presented a few emergency messaging system proposals made by other researchers and also introduced a new proposition of our own. We have given a brief survey of the delivery of emergency alerts using the Internet. We have outlined a structure for communication between server and client. This is the most important part of a messaging system, when further research and demo application development are included in the scope. The proposed system uses Atom feeds to deliver alert messages, which makes it possible for clients to use any kind of device, from mobile terminals to desktop computers. The next step in our research will be to develop a demo of both the server and the client application.

REFERENCES

1. Alexander, D. (2002) Principles of Emergency Planning and Management, Terra Publishing.
2. Botterell, A. (2003) An Advanced EAS Relay Network Using the Common Alerting Protocol, White Paper, http://www.incident.com/cap/docs/aps/Advanced_EAS_Concept.pdf (retrieved 10.12.2008)

3. Botterell, A. (2006) The Common Alerting Protocol: An open Standard for Alerting, Warning and Notification, in proceedings of the 3rd International ISCRAM Conference (B. Van de Walle and M.Turoff, eds.), Newark, NJ.
4. CAP Cookbook (2008) http://www.incident.com/cap (retrieved 10.12.2008)
5. Common Alerting Protocol v1.1 (2005) OASIS Emergency Management TC, http://www.oasisopen.org/committees/download.php/14759/emergency-CAPv1.1.pdf (retrieved 10.12.2008)
6. Emergency Data Exchange Language, Distribution Element, v1.0 (2006) OASIS Emergency Management TC, http://docs.oasis-open.org/emergency/edxl-de/v1.0 (retrieved 10.12.2008)
7. Finnish Funding Agency for Technology and Innovation (Tekes), http://www.tekes.fi/eng (retrieved 15.4.2008)
8. IBM developerWorks (2007) X-Capatom, https://www.ibm.com/developerworks/library/x-capatom (retrieved 10.12.2008)
9. NICT (2007a), in proceedings of the First International Symposium on Universal Communication (ISUC), Kyoto, Japan.
10. NICT (2007b), in proceedings of the First International Workshop on Knowledge Cluster Systems. Keijanna, Japan.
11. Organization for the Advancement of Structured Information Standards (2008) http://www.oasis-open.org (retrieved 10.12.2008)
12. Organization for the Advancement of Structured Information Standards (2005) User Requirements Document OASIS_TA22_REQ_003_DSF
13. RFC 4287 (2005) The Atom Syndication Format, IETF Network Working Group
14. RFC 5023 (2007) The Atom Publishing Protocol, IETF Network Working Group
15. Soini, J., Leppäniemi, J. and Jaakkola, H. (2008) Towards Seamless Collaboration in Distributed Disaster Knowledge Management, in proceedings of the Information Society 2008 Conference, Ljubljana, Slovenia.
16. Tampere University of Technology (TUT), http://www.tut.fi (retrieved 15.4.2008)
17. Verma, P. and Verma, D. (2005) Internet Emergency Alert System, in proceedings of Military Communications Conference (MILCOM 2005), Hawthorne, NY, USA.
18. Westfall, J. (2007) CAP Index Format Proposal, http://www.incident.com/cap/feed-format-proposal.pdf (retrieved 10.12.2008)

Publication II

Rantanen P., Sillberg, P., Jaakkola, H. and Nakanishi, T., "An Asynchronous Message-based Knowledge Communication in a Ubiquitous Environment", Database Systems for Advanced Applications, Springer, ISBN 978-3-642-14588-9, DOI 10.1007/978-3-642-14589-6_44, pp. 434-444, 2010.

Original publication is available at `http://link.springer.com/chapter/10.1007%2F978-3-642-14589-6_44`.

An Asynchronous Message-based Knowledge Communication in a Ubiquitous Environment

Petri Rantanen, Pekka Sillberg, Hannu Jaakkola, and Takafumi Nakanishi

[1] Tampere University of Technology (TUT),
Pohjoisranta 11 A, 28100 Pori, Finland
{petri.rantanen,pekka.sillberg,hannu.jaakkola}@tut.fi
[2] National Institute of Information and Communications Technology (NICT),
3-5 Hikaridai, Seika-cho, Soraku-gun, Kyoto, 619-0289, Japan
takafumi@nict.go.jp

Abstract. This paper presents the required operational logic for relaying user requests from traditional Server/Client-based systems to services or additional information sources that require asynchronous communications. The paper describes a simple syntax for user-generated messages that can be used to determine where the messages should be forwarded. The paper also explains how replies to these messages should be processed. In the scope of this paper the Knowledge Grid works as the external source of information, but the same principle could be applied to any information source.

Key words: IP-based alert message system, Knowledge grid, Message-based System, Asynchronous communication

1 Introduction

The original context of the topic discussed in this paper is management of disaster related knowledge in connection with serious accidents and catastrophes. In this kind of situation the availability of up-to-date information about the accident is important for the authorities as well as their opportunity to guide people in the accident area to survive in exceptional circumstances. A fast reaction to the situational knowledge is a precondition for successful rescue operations. It provides an opportunity to limit financial losses and human suffering. In the globalized world, major accidents and catastrophes are also becoming global – involving citizens of several countries. As a result, the interest in rescue operations and information distribution is also global. In many cases even the access to public knowledge – available on the Internet – is highly beneficial.

This paper is based on the long-term collaboration between the organizations of the authors – Tampere University of Technology (Finland), and NICT and Keio University (Japan). In addition, the research consortium covers partners from Germany, the Czech Republic, and Indonesia. The purpose of the joint research activity is to develop technologies and processes that improve the availability of knowledge in disasters. The kernel of this joint activity is the GRID

based distributed knowledge management platform. This system has the capacity to mine information items from public sources and authorities repositories, and combine individual information items to create valid knowledge to be utilized in the disaster context. Seamless – place, time, terminal independent, symmetric – access to the sources of knowledge is recognized as vital. This is supported by the Asynchronous Message-based Communication system that is being developed in this joint research activity. The system is based on open protocols and provides two-directional access between the terminals available in the disaster area and the services provided by the connection server. The upward direction may be utilized to transfer data, e.g. photos, data stream or measured data, from the accident area to the authorities, and the downward direction may be utilized to distribute information and guidance to the people in the accident area.

The wider context of the project related to this paper is the openness and interoperability of information systems based on the ideas of the Service Oriented Architecture (SOA) approach. It is based on standardized interfaces and protocols and deep understanding of complex processes crossing organizational borders. The inspiration for the project comes from the fact that every serious accident and catastrophe demands coordinated collaboration between several authorities that may also represent different nations and cultures. The knowledge repositories of authorities – legacy systems – are typically closed, unable to interoperate with each other and with open sources of knowledge. SOA-based interfaces would provide a safe and easily implementable solution to this problem. The Asynchronous Message-based communication protocol further extends access to registered service users using their mobile terminals to communicate with the integrated service architecture.

This paper introduces the results of the joint Finnish-Japanese research activity. The focus of the paper is to describe the integration of the systems designed and developed by the organizations of this paper's authors. The purpose of the integration is to promote the use of the both systems and act as a starting point for developing more features between the systems. Chapter 2 opens the discussion on the motivation to develop asynchronous message-based communication. Chapter 3 describes the IP-based message delivery system developed in the earlier research. Chapter 4 gives an overview of the main components of the extended system, Chapter 5 explains the operational logic of the system, and Chapter 6 includes the summary of the research.

2 Why Asynchronous Message-based Communication?

The nature of a Knowledge Grid query is that it is unknown beforehand how long it will take. Processing the result may take any amount of time from a few seconds to a very long time. Opening and keeping the request open to the service provider until the result is received is not very efficient and ties up resources for the whole length of the request. There are a few ways to build asynchronous Web services to work around this issue. All listed methods have their own advantages and disadvantages. [10]

One of the approaches is to use one-way notifications between client and service provider. Also a few variations of request/reply operations can be used. The request/reply operations with polling requires the most simple client implementation as it does not need its own Web service but requires the client to implement the polling system to check the service provider periodically for results. It may also require more than one request to retrieve the response if the service provider has not yet completed the query. This approach provides a level of decoupling but leaves the responsibility of retrieving the results to the client.

As the processing time of the Knowledge Grid query may be unknown, it might be better to let the service provider handle the retrieval and sending of results. Both one-way notification and a simple request/reply operation can be used. The client is required to implement its own Web service, supply the endpoint address of this service so the service provider can send the results and supply correlation ID to identify the query. The disadvantages of this type of approach are that it requires quite a large amount of implementation. Also there may be difficulties for the service provider to reach the client when it needs to send the results. On the other hand, if the client is already acting as a Web service for someone else, typically most of the implementation, firewall configurations and so on have already been done. The advantages are a high level of decoupling and the fact that the client can just "fire and forget" the query and let the service provider handle the sending of the results.

One of the possible scenarios for the system could be a traveler or tourist, who uses the message delivery system for receiving travel news, advisories and alerts. When the traveler encounters some event, he or she may want to report that event to other users of the system. Also our traveler may want to know more detailed information on this kind of event. In this case a new Knowledge Grid query may be initiated. The data provided by the Knowledge Grid is background data on the requested subject (and not for example situational awareness data for an emergency), and as such a small delay in response is not critical. The client software may also poll the feeds at some predefined time interval, which means the information will not always be retrieved in real-time in either case.

3 IP-based Knowledge Delivery System

The system described in this paper is based on the research published in paper [2]. The paper proposes a system architecture that can be used to provide alert messages to mobile and desktop clients using IP-based networks. The system uses a traditional Client/Server architecture and secure server approach [1]. The system offers two alternative messaging channels. The first messaging channel is an SOAP-based [4] synchronous bi-directional communication between the server and the client. The other channel is an Atom feed-based [3] information resource that can be retrieved by any kind of client from proprietary custom software to basic Really Simple Syndication (RSS) readers.

The fundamental idea of the system has been unaltered, but it has received more functionality. The original system concept used RSS-feeds and other in-

formation sources provided by trusted parties as resources. Messages from these sources were added to Atom feeds provided by the server or sent directly to clients depending on the use case. The system also offered the possibility for clients to add additional information to existing feeds. In this paper the approach is extended by adding a Knowledge Grid as an additional resource. The server has been added with the ability to process and respond to client generated messages and to relay clients requests to server-known information sources. In this paper, the designed server is called Message Server.

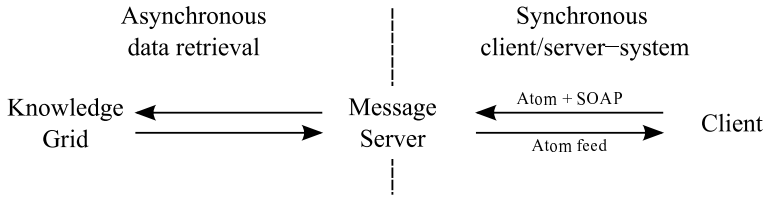

Fig. 1. Communication overview.

Figure 1 shows the Server/Client-system with the Knowledge Grid working as an additional information resource. The right side of the figure shows the communication between the clients and the Message Server. The feeds are pulled by the clients at some pre-defined time interval using standard HTTP GET commands and SOAP-messages are transferred using HTTP POST. In our use cases, the clients are users, but they could as well be automated sensors or any other kind of devices.

The original system proposal uses Common Alerting Protocol (CAP) [8] encapsulated in SOAP messages. CAP-messages are a standardized way for informing and alerting in emergency situations, but may be cumbersome for more casual messaging. Therefore we added a GeoRSS [5] extended Atom as a new message type. Though GeoRSS has multiple variations [6][7] and is not an official standard, we chose it because it is often used on the Internet and has a standard way of presenting coordinate data [9]. For our experiments, GeoRSS is a good starting point for testing of the system. Adding different kind of message types should be quite easy based on our experience of adding the GeoRSS message type. For more specific cases, a suitable message type should be created and used for better communication and messaging.

The left side shows the asynchronous communication between the Message Server and the Knowledge Grid. The communication is shown in the figure below to give a complete overview of the designed system and is explained in more detail in the following chapter.

4 Asynchronous Communication for a Knowledge-based System

One of the design goals of accessing Knowledge Grid was to clearly separate the services of the Knowledge Grid and the Message Server. This approach has the benefit of requiring both of the systems – Knowledge Grid and Message Server – to implement only the common interface for communication. The systems can be developed independently, and if needed the designed interfaces can be opened to provide additional services to third party clients and servers.

Figure 2 shows an overview of asynchronous communication. The Knowledge Grid and the implemented Grid Access Gateway which is needed to enable communications with the Knowledge Grid and Message Server are shown in the figure. These three components will be explained in more detail in the following section. In our implementation, the Knowledge Grid functioned as the external information source, but the same operational logic could be applied to any source including possible third party systems.

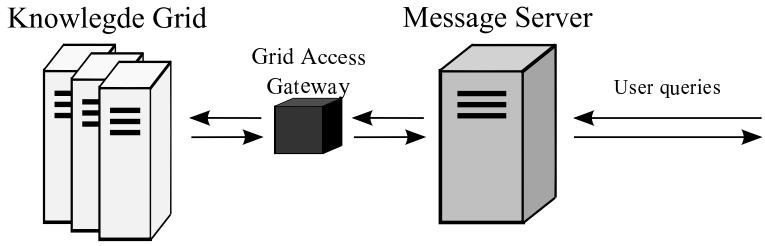

Fig. 2. Asynchronous communication.

4.1 Knowledge Grid

Recently, a wide variety of knowledge items have been created by using collaborative environments in each community. Unfortunately, this knowledge is mainly utilized only for sharing information resources within each community.

An event affects various aspects of an area, field, or community. For example, in the case of a disaster, a secondary impact and secondary disaster may affect other areas such as the environment, economy, and healthcare. In order to understand the arbitrary concept, it is important to transfer significant knowledge related to accidental or irregular events to actual users from various knowledge bases.

References [11–13] have proposed a Knowledge Grid, which provides an infrastructure of acquisition, analysis, and delivery for knowledge sites. In particular, the interconnection method for knowledge bases has been presented by

references [12, 13]. This method provides the interconnection of heterogeneous knowledge based on the Knowledge Grid and representation of related concepts in heterogeneous fields. It navigates to related contents in these heterogeneous fields depending on the context. Furthermore, these knowledge bases that exist independently can be used more effectively when arranged.

This method represents a lot of different concepts with contents over the heterogeneous fields in a PC client. However, in the case of mobile devices, it is difficult to represent these related contents provided by the method, because a mobile device does not have a screen large enough to represent a lot of various related contents and the mobile user cannot wait for the computation of this method.

In order to solve these issues, it is important to realize asynchronization communications and the resulting delivery by streaming. It is necessary for mobile users to push the halfway result at any time to deliver the results which have been computed by deep analysis during a long processing time. A mobile device can receive the halfway result at any time and it can be presented to the user little by little. In this paper, we have developed asynchronized communication for the interconnection computation of heterogeneous knowledge based on the Knowledge Grid.

4.2 Grid Access Gateway

The Grid Access Gateway works as a common interface between the Message Server and the Knowledge Grid. The Message Server and the Knowledge Grid did not have mutually compatible interfaces, which meant that the design of a middleware between the two systems was necessary. One option would have been to change the interface of either the Knowledge Grid or the Message Server to include compatibility with the other system, but we decided to couple the systems as loosely as possible. This approach allowed the gateway to be developed separately, which also means that the gateway itself would work like any third party system connected to a server.

In its current implementation, the only function of the gateway is to provide simple message translation services. This means converting the messages forwarded by the Message Server to the format understood by the Knowledge Grid and vice versa. The Knowledge Grid messages – or more precisely replies to the forwarded user queries – are converted to the client message format and sent to the Message Server like any other message. In other words, the Grid Access Gateway works like an additional service for the server and like a basic client software.

4.3 Message Server

The original Message Server implementation functioned like a traditional web server and used basic transactions when transferring information to and from the clients. In a strict alert message relay this approach poses no problems, but by itself it does not include internal logic to handle asynchronous data transfers.

In the case of an external information source that for some reason cannot send immediate responses to requests this causes a problem. Our solution was not to change the behavior of the Message Server, but to add to it a functionality to recognize and forward to other services user queries that included messages needing additional processing. This way the server could be kept as simple as possible, but still enable enhanced functionality to clients using the server.

If the chosen design is used, there is no need for a separate server interface for the responses of the forwarded queries. The receivers of the relayed requests can work as "clients" to the Message Server and send the responses as a normal message. This message is then processed normally by the server and added to a feed which can be read by other clients or forwarded to clients when needed. Depending on the case, the messages can also be sent directly to clients by the service that processed the request. The latter case will save server resources, but on the other hand it may be easier for the users if they receive all the messages from the same source, and that all the services seem to be offered by the same party.

Relaying the requests also simplifies the addition of new services to the Message Server as it is not necessary to modify the server to understand all the possible different kinds of service responses. The downside of this approach is that the responses that can be added to the servers feeds are limited to the message types recognized by the server. For text-based information relay this should not pose a problem, but for more complex data types – like video or audio – a different approach must be taken. One possible solution could be to provide a link to the data and host the content on another location, which would also reduce the network load on the Message Server. Another option would be to encode the data inside the XML using for example Multipurpose Internet Mail Extensions (MIME).

5 Implementation and Usage Scenarios

In this implementation, encryption is not used by default. Encryption can be enabled if required by the user, but as long as the messages are considered to be casual user-to-user messages, encryption is not necessary. One possible encryption method is for example Transport Layer Security/Secure Socket Layer (TLS/SSL).

Accessing the data in external sources requires the user to type in the query by using special markup recognized by the Message Server. The query can be written to any message type (for example CAP or GeoRSS) and to any element that is allowed by the type to have a clear text description. We chose to use regular expressions to match and extract the query from the message. This gives the possibility to define the query syntax flexibly. The following regular expression is the currently used in our implementation:

Regular expression for matching and extracting the query

```
(\w|\-)+(\?\?)(\w|\s)+(\,(\w|\s)+)*\.
```

The syntax consists of five sections: target, double question marks, method, parameter(s), and end sign. Basically target, method and parameter consist of alphanumeric characters. In addition to this, the target may consist of character "-". The method and the parameter may additionally have whitespace characters. The target is always followed by double question marks. The target method and parameters are separated with a comma, and the end sign is a period. An example of a external query shown below:

Example syntax for invoking external query

```
nict-kcs??article_query,EnvironmentNews,Tokyo,Hurricane.
```

Target in this example is "nict-kcs" and method is "article_query". After the method there is three more parameters: context, place and keyword and values are "EnvironmentNews", "Tokyo" and "Hurricane" respectively. Interpretation of the query would be that user wants NICT's Knowledge Grid to provide some kind of information (articles) about hurricanes around the Tokyo area in the context of environment news.

To reduce the need for memorizing the full syntax, the user does not need to fill in all the parameter(s) and can let the client or Message Server software fill in the missing parameters. For example, if the full method call requires a target, target method and parameters which are context, datetime, place and keyword, the Server software could accept queries with only target, place and keyword. The remainder of the required parameters would be filled in automatically using default values. How and which parameters are requested from the user depends on the service that needs to be invoked and on the client software used.

Based on the given target and the function, the Message Server can relay the request to the actual service provider. The server itself does not process the parameters. This way the server only needs a list of valid targets and their real location, for example the IP address and the function that should be called on the target. Based on the given information the server creates a simple SOAP-encapsulated message targeted to the requested function. The message includes the unprocessed parameters, unique identifier, URI, function and XML namespace information. The identifier can be used by the server to match service responses to specific requests. The servers URI, function and namespace used by the XML are provided to services in order to make returning the results possible. Just like the messages between the server and the clients, the messages between the server and the service providers can be encrypted when required.

5.1 Usage Scenarios

Information retrieval from the Knowledge Grid is divided into two phases, creating the Knowledge Grid query and receiving the results. Figure 3 shows a simplified sequence diagram of creating a Knowledge Grid query. In fact, the Knowledge Grid and the Grid Access Gateway shown in the figure are interchangeable as the user-made query can be directed to basically any other kind of service. These use cases are applying the Knowledge Grid and the Article

query as an example. The users query is basically a query for the articles that are related to the context and keywords that the user is interested in, hence the terms "Article query" and "Article feed" which contains the results of the query.

The use case begins when the user creates a *New message*. This new message is then added to the Message Server's own feed database and can then be read by other users. If the message contains the query syntax described in section 5, the Message Server will then generate a corresponding query to the target specified by the user. In this case, this is the *Article query* in the figure. The target of the query, the Knowledge Grid, has a Web service interface to provide the article information. This is the target where the query should be sent by the Message Server. When the Grid Access Gateway receives the query, it will create a *Grid query* which starts processing on the Knowledge Grid.

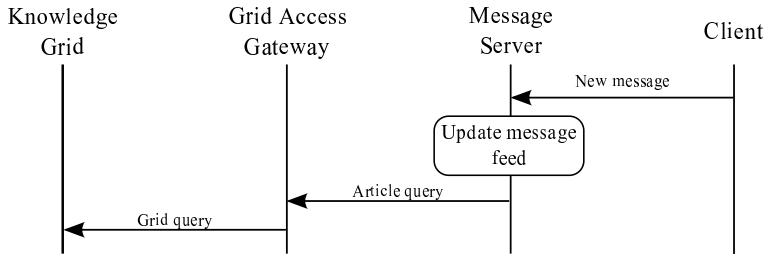

Fig. 3. Creating a new Knowledge Grid query.

The second use case – receiving the results of the Knowledge Grid query – is shown in Figure 4. The Grid Access Gateway periodically checks for the status of the grid query and, if the Knowledge Grid has processed the results, they will be returned to the Grid Access Gateway. After this, the Grid Access Gateway will add identification data to the result and send it back to the original caller. The original caller may then process the information received, like in our example, the update article feed. This phase will be repeated until the Knowledge Grid announces to the Grid Access Gateway that there will be no more results.

Users may read the article feed at any point after the initial article query has been made. At first the feed will be empty, but once the Knowledge Grid has processed and returned some results, the feed will be updated and the items in it will increase.

6 Summary

This paper presented a simple operational logic that allows connecting a synchronous Client/Server system with an additional information source or service that requires asynchronous communications. The paper described the required

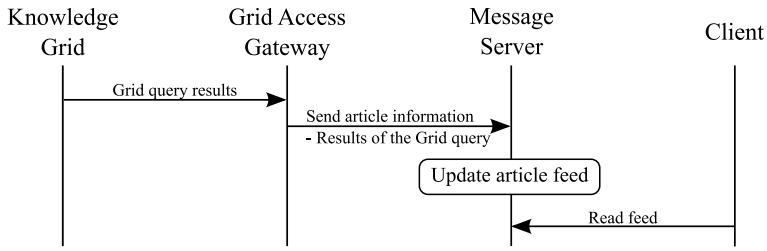

Fig. 4. Receiving the results of the Knowledge Grid query.

changes on the server side and the use cases involved in forwarding messages from user queries to the Knowledge Grid and returning the replies to the users. In the designed system, the Grid Access Gateway was used to transform message formats between the Message Server and the Knowledge Grid.

In future research, the designed system – including the Message Server, Knowledge Grid and client software – will be further modified to handle a wider range of message types. The research will also concentrate more on general messaging and less on pure alert message delivery. This will include researching the presentation of Knowledge Grid data to an end user who is using a mobile device.

Acknowledgments.

This work is partially funded by the Finnish Funding Agency for Technology and Innovation (Tekes) Seamless Services and Mobile Connectivity in Disaster Knowledge Management (SSMC/DDKM) research project and the Academy of Finland UbiKnowS project. In SSCM/DDKM the work is based on the memorandum of understanding – "mobile knowledge management architecture" between Tampere University of Technology (TUT) and the National Institute of Information and Communications Technology (NICT).

References

1. Botterell, A.: An Advanced EAS Relay Network Using the Common Alerting Protocol, White Paper (2003)
2. Sillberg, P., Rantanen, P., Saari, M., Leppäniemi, J., Soini, J. and Jaakkola, H.: Towards an IP Based Alert Message Delivery System, In: Information Systems for Crisis Response and Management Conference, Gothenburg, Sweden (2009)
3. The Atom Syndication Format, RFC 4287, http://www.ietf.org/rfc/rfc4287.txt
4. SOAP Version 1.2, W3C Recommendation 27 April 2007, http://www.w3.org/TR/soap12
5. GeoRSS Wiki, http://www.georss.org/
6. GeoRSS encodings, GeoRSS Wiki, http://www.georss.org/Encodings
7. W3C Geospatial Vocabularity, W3C Incubator Group Report 23 October 2007, http://www.w3.org/2005/Incubator/geo/XGR-geo-20071023/

8. Common Alerting Protocol v1.1, OASIS Emergency Management TC, http://www.oasisopen.org/committees/download.php/14759/emergency-CAPv1.1.pdf
9. World Geodetic System 84 (WGS 84), Implementation manual, version 2.4, World Geodetic System (1998)
10. IBM Developer Works, Asynchronous operations and Web services, Part 2, http://www.ibm.com/developerworks/webservices/library/ws-asynch2
11. Zettsu, K., Nakanishi, T., Iwazume, M., Kidawara, Y. and Kiyoki, Y.: Knowledge Cluster Systems for Knowledge Sharing, Analysis and Delivery among Remote Sites, Information Modelling and Knowledge Bases (IOS Press), 19, pp.282-289, 2008.
12. Nakanishi, T. , Zettsu, K. , Kidawara, Y. and Kiyoki, Y.: Towards Interconnective Knowledge Sharing and Provision for Disaster Information Systems -Approaching to Sidoarjo Mudflow Disaster in Indonesia-,In Proc. of The 3rd Information and Communication Technology Seminar (ICTS2007), pp. 332-339, Surabaya, Indonesia (2007).
13. Nakanishi, T. , Zettsu, K. , Kidawara, Y. and Kiyoki, Y.: Approaching to Interconnection of Heterogeneous Knowledge Bases on a Knowledge Grid, In Proc. of The International Conference on Semantics, Knowledge and Grid (SKG 2008), pp. 71-78, Beijing, China (2008).

Publication III

Sillberg, P., Kurabayashi, S., **Rantanen P.** and Yoshida, N., "A model of evaluation: computational performance and usability benchmarks on video stream context analysis", Information Modelling and Knowledge Bases XXIV, IOS Press (Frontiers in Artificial Intelligence and Applications; Volume 251), ISBN 978-1-61499-176-2, DOI 10.3233/978-1-61499-177-9-188, pp. 188-200, 2013.

Original publication is available at `http://ebooks.iospress.nl/publication/7645`.

A Model of Evaluation: Computational Performance and Usability Benchmarks on Video Stream Context Analysis

Pekka SILLBERG [a], Shuichi KURABAYASHI [b], Petri RANTANEN [a] and Naofumi YOSHIDA [c]

[a] *Tampere University of Technology (TUT), Pori, Finland*
[b] *Keio University, Japan*
[c] *Komazawa University, Japan*

Abstract. This paper presents a model of evaluation for computing performance and usability benchmarks. The target evaluation is context analysis of video streams. The paper describes the required system components, system architecture and evaluation model, as well as the test cases that can be performed using the designed environment. During the modelling, design, implementation and preliminary tests of the environment, some issues, problems and good-to-know key points were discovered which are explained in this paper. Reference guidance for future test cases is also given. The modelling and experimental results presented in this paper are based on the test environment, and the observations of the cooperation project between Tampere University of Technology Pori Unit and Keio University Shonan Fujisawa Campus.

Keywords. Kansei, Video Analysis, Test Environment, Evaluation

1. Introduction

The possible problems caused by the classification, search and analysis of the vast amount of information available in both offline and online sources (web) are one of the research issues in the on-going MOP (Mukautettavat OhjelmistoPalvelut, in English: Adaptive Software Services) project co-ordinated by Tampere University of Technology (TUT) Pori Unit. An important part of the research is the study of methods that realize efficient and automatic generation of meaningful (usable) metadata. One of the aforementioned methods is the Kansei-based analysis designed by Kiyoki et al. [9][10]. This method – called the Mathematical Model of Meaning (MMM) – enables the creation of word matrixes that can be used to calculate the correlations between words, and also includes the means for extracting (Kansei) words from various media content (video [18], audio [7][6], web [8]). TUT Pori Unit and Keio University have a long history of cooperation, and in the spirit of previous good experiences a researcher exchange was organized, in which two researcher from TUT Pori Unit (the authors of this paper) travelled to Keio SFC (Japan) to study the MMM and its applications.

1.1. Project Background

The MMM has been proved as working concept by many previous papers and research projects (see Chapter 1), but no uniform test environment exists for running more comprehensive tests and performance benchmarks. The researchers at Keio SFC expressed great interest in testing the MMM inside a more complete system with real-life data. In addition, the industry partners of the MOP project were very interested in the video analysis features promised by the MMM. As a result of these wishes, a test environment consisting of a simple test application, web service and video analysis back-end were developed where the main focus was laid on video analysis. The goal was to provide a system that could be used to analyze videos, and the system should also allow the end-user to search for the analyzed video content using a keyword-based search.

1.2. Kansei-based Video Analysis

In the scope of this paper *Kansei*[1] is limited to the context of video analysis, and more specifically to the video analysis methods provided by the MMM. The MMM uses color histograms to define the meanings of the Kansei words, which in this case are adjectives. The words and their relations to certain color patterns (histograms) are defined by the research done in the field of color psychology. In our test environment a color impression matrix containing 182 adjectives, each corresponding to a specific emotion in human perception, was used.

Figure 1. Color-Emotion association for defining color features in 182 sets of color schema related to 130 color variations. Each color-emotion definition, such as *vigorous(cs1)* defines a set of weight for colors.

Figure 1 shows example color adjectives that define color features in 182 sets of color schema related to 130 color variations. Each color schema corresponds to a specific emotion. This matrix consists of 120 chromatic colors and 10 monochrome colors defined in *Color Image Scale* [11] which is based on the Munsell color system. Each color schema, which corresponds to a specific emotional perception of humans, is described by using a combination of several colors from the 130 basic color set. The more in depth explanations for the color-to-emotion relationship in the context of image and video analysis can be found in the papers by Sasa et al. [22] and Kurabayashi et al. [13]. The purpose of the method is to generate a *feeling* or *mood* automatically for the media content. For example, based on the analysis a video might be classified by the keyword *aggressive* or by the keyword *cute*. The method also calculates values for each keyword,

[1] Kansei Engineering was invented by Mitsuo Nagamachi in the 1970ies [14].

which can be used for comparing the media content [12]. The values indicate how similar to the defined histograms the analyzed video content was. This functionality could be used to rank the search results (for example, by showing better matches at the top of the result list) based on how well they match the keywords provided by the user. An important note to make is that the video analysis is based on the picture alone, the audio track is ignored.

The system analyzes the continuous color-emotion features in a video by performing the following four steps as depicted in Figure 2.

- Step-1: The system decodes an input video stream and captures each frame in the decoded video stream.
- Step-2: The system converts RGB (Red, Green, Blue) color values to HSV (Hue, Saturation, Value) color values per pixel of each image. HSV is a widely adopted color space in image and video retrieval because it describes perceptual and scalable color relationships.
- Step-3: The system clusters HSV pixels into the closest color of the predefined 130 Munsell basic colors, and calculates the percentage of each color to all pixels of the image. And then the system creates 130 HSV color histogram.
- Step-4: The system extracts the color-impression for each frame in the video by correlation calculations between 182 color schemas (182 impression word sets) and 130 basic HSV colors.

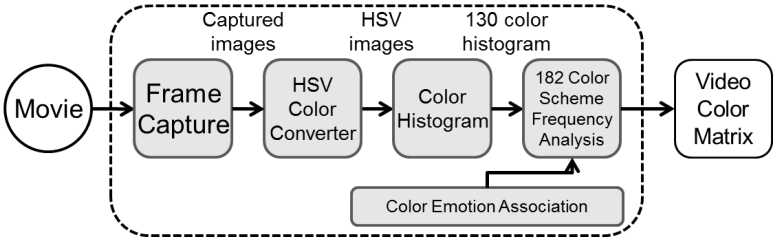

Figure 2. A procedure for analyzing color-impression along timeline in a video data.

2. Test Environment

Figure 3 illustrates the overall system architecture. The system consists of three components (Video Analysis Back-end, Web Service, and Test Application), which two of (Video Analysis Back-end and Web Service) can be located on the same hardware if so desired. The reasoning behind separating the Video Analysis Back-end and the Web Service is to provide better scalability (the possibility of addding additional back-end analyzers) and also to enable the testing of the back-end implementation with or without the Web Service. If the Media Analysis Back-end and Web Service had been integrated into a single server, that would have made it more difficult to benchmark the computing performance of the individual components of the system. In addition, the interfaces provided by both the Video Analysis Back-end and Web Service were implemented using

Representational State Transfer (REST) architecture. The use of RESTful web service methods allowed the initial tests to be performed with a web browser, without the need to develop the Test Application immediately.

All communications between the Test Application and Web Service were implemented using Hypertext Transfer Protocol (HTTP) and Extensible Markup Language (XML). The Video-Analysis Back-end used in our test environment provides the response messages in JavaScript Object Notation (JSON). There have been many studies on the network performance with comparisons of XML and JSON [20][19][3], and as a general rule JSON is slightly more efficient. In the case of our test environment, the choice of language for data representation, formatting and interchange can be considered less important. There are two reasons for this. Firstly, in most cases (testing the behavior on different video codecs, testing different video content, etc.) all results are achieved running the same system, in which case the performance effect caused by the language is always the same. Secondly, the time used for basic data transfers (HTTP GET/POST), and parsing the associated content is negligible when compared to the time taken to download and analyze the video files. In the light of the above, the choice was made to use the languages that would allow the easiest and fastest implementation of the system. Our Video-Analysis Back-end implementation (see subsection 2.1) already had a partial interface implementation using JSON, and the use of XML was recommended for the platform we used to implement the Web Service. Another option would have been to use custom-made protocols or languages for representing and transferring the data, but the design, implementation and testing of the required components is a non-trivial task, and the decision was made to use well-known solutions.

In the test environment, the Test Application works both as the provider of the video content and as the end-user of the system. In practice, the content could be provided by the user or by some third party content provider.

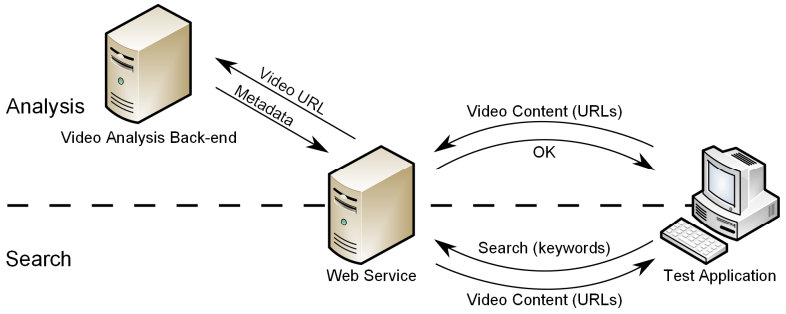

Figure 3. Test environment overview.

The Figure 3 also shows the two main use cases for the performance analysis. The upper part of figure shows the phases required for analyzing the video content and the lower part shows the phases needed for content search. In more details the phases are as follows:

- Video analysis (upper part of Figure 3):

- The video content (a single Uniform Resource Locator pointing to the video, or a list with multiple URLs) is sent from the test application to the Web Service.
- The content is passed by the Web Service to the Video Analysis Back-end.
- The Video-Analysis Back-end downloads the actual video file from the given URL and analyzes the content. The results (the matching words and their values) are returned to the Web Service.
- The Web Service stores the results as a metadata related to the given Video Content on its local database.
- The Web Service notifies the Test Application of the results of the video analysis.

- Search based on keywords (lower part of Figure 3):
  - The Test Application sends the keyword (or list of keywords) to the Web Service.
  - If videos matching the submitted keywords are found, the results (the list of video URLs and their corresponding keyword/value pairs) are returned to the Test Application.

As mentioned before, using the test environment, the Test Application could directly send the Video Content directly to the Video Analysis Back-End, but in that case the results would not be saved by the Web Service, and thus the results could not be included in the results list returned in the search case.

2.1. Video Analysis Back-end

The Video Analysis Back-end (in Figure 3) is implemented using the MediaMatrix [12][16] software developed at Keio SFC. MediaMatrix includes a REST interface that can be used to analyze video content, and also provides the required functionality to customize the color impression matrix used – in practice by making it possible to upload new color definitions in the form of comma-separated values (.csv) file. The importance of allowing the use of multiple color definitions is crucial because people's association of colors to emotions varies greatly. In addition to personal differences, there are cultural and contextual differences that need to be taken into account. For example, in western cultures black is often associated with mourning, but in Japanese culture it can mean age and experience. For these reasons, the choice of color definitions should be made very carefully in the end-user products, and preferably tested with user studies. The current implementation for the Video Analysis Back-end processes one frame of video per second without any additional optimizations. The frame extraction and the decoding of the video content are done using FFmpeg [4], which allows the use of a wide range of video formats. The MediaMatrix also enables the analysis of video content using three possible methods: finding words that are *dominant* throughout the entire video; finding words that appear *frequently* during the entire video; and finding *emergent* words. Emergent words have considerably higher values than other words whether they appear consistently in the whole video or just within a shorter time-span.

2.1.1. Video Analysis, Example Method Call

An example method call to the REST interface provided by MediaMatrix can be seen below:

```
Example Query (Video Analysis):
http://video-analysis.example.org/MediaMatrixService/analyzeURL?url=
http%3A%2F%2Fexample.org%2video.mp4

Example response in JSON format:
{
   "MD5":"07097427c5aa85620232b836ccc409aa",
   "frequent":{
      "intense":2.134337902069092,
      ...
},
   "dominant":{ ... },
   "emergent":{ ... }
}
```

In this example the video content has been provided by passing a single URL as the parameter. The response contains the MD5 Message-Digest Algorithm [5] of the color impression matrix used for generating the metadata for possible future reference. The other included information are keyword/value pairs grouped by the method that was used to extract them. For the sake of simplicity, the list of keyword/value pairs has been reduced to a single pair and only the results generated by the *frequent* method are shown. The actual results usually contain a predefined number of keyword/value pairs (for example, the top 10 keywords extracted by each method). The result could include all extracted keywords, but keywords with very low values (partial matches) may not be meaningful. The word values are normalized from the original matrix used by the MMM algorithms. The values inside each method (*frequent, dominant, emergent*) are comparable within the group, but not necessarily with values located in other groups.

2.2. Web Service

The main purpose of the Web Service (server) in Figure 3 is to provide a database that can be used later for retrieval of the video metadata previously generated by video analysis. In our test environment the MediaFusion platform [15] was used. This platform offered built-in tools for easily implementation of the REST interfaces needed by the Test Application and the communication protocols used to transmit the video content to the Video Analysis Back-end. The role of the Web Service is fairly simple, and basically any platform, from professional server environments to open-source implementations, could be used. When implementing the Web Service the platform most familiar to the developers often offers the easiest and the fastest way for design and implementation.

Using the interfaces provided by the Web Service, the Test Application can perform keyword-based queries, which can include any number of keywords. For example, *aggressive forceful* would return all videos that contain aggressive and forceful content). In addition to keywords the system should accept values for the keywords. This functionality could be used for both limiting the search results (only return video content that is close to the values) and also for providing a similarity search functionality. In the case of a similarity search, the keyword/value pairs for the desired content must be known beforehand. The pairs could have been received from a previously executed keyword search or from a manual run of a video through the Video Analysis Back-end.

Another possible implementation for the similarity search could be to allow the user to submit a reference video that would be analyzed by the Video Analysis Back-end and the results of that analysis would be used as the keyword/value pairs for the search. If the aforementioned implementation is desired, it should be noted that based on the preliminary computing performance tests the current implementation of the Video Anal-

ysis Back-end is somewhat resource-intensive. The analysis ratio (how many minutes of video can be analyzed in one minute) is currently about 10:1 for basic web content (360p, 600kbit/s), but drops closer to 1:1 for high definition content (1080p, 15Mbit/s). In practice, low quality user content could be analyzed on the fly, but the wait times for high quality content analysis would be too long.

Besides the web service (REST) interfaces provided for the Test Application and the required functionality to enable the method calls to the Video Analysis Back-end, the Web Service includes a database for storing the results of the video analysis. The database should at least contain tables for the metadata required for video searches (video URL and the related keyword/value pairs). In addition, the method used to extract the keywords (frequent, dominant or emergent) and the information about the color impression matrix used can be helpful, especially since the metadata generated using different methods and matrixes will not be comparable.

2.2.1. Video Search, Example Method Call

An example method call to the keyword-based search REST interface is seen below:

```
Example Query (Keyword Search):
http://webservice.example.org/rest/kanseiservice/patternSearch?pattern=intense

Example response in XML format:
<?xml version="1.0" encoding="UTF-8" ?>
<response service="kanseiservice" method="patternSearch">
  <stat>ok</stat>
  <results>
    <result>
      <itemId>0</itemId>
      <url>http://example.org/video.mp4</url>
      <keywords>
        <keyword>
          <method>frequent</method>
          <value>2.134337902069092</value>
          <word>intense</word>
        </keyword>
        ...
      </keywords>
    </result>
  </results>
</response>
```

The list of keywords, or in this case the single keyword *intense*, is provided by using the parameter *pattern*. The return message includes the list of results that in the example case consists only of a single result, and the list of keywords, keyword values and the information about which method was used to extract the keyword (in the example the method was *frequent*). The element *itemId* is a unique identifier that can be manually set as a parameter for the video URL when the URL is analyzed. If it is omitted it will be generated automatically by the Web Service as long as the analysis is executed through the interface provided by the Web Service, and not by calling the interface offered by the Video Analysis Back-end directly.

2.3. Test Application

The Test Application (in Figure 3) was implemented with the Qt framework [21]. Qt was chosen because of its robust and easy to use (C++ -based) selection of network and user interface components that can be used in a cross-platform environment (Windows, Mac OS X, Linux/Unix support for desktops, Symbian support for mobile devices). The

selection of programming language for the application is not crucial, and a comparable language for Android or iOS could equally well have been used. The main feature requirements were a simple user interface and the ability to measure the time spent running the designated computing performance benchmarks. Both features are easy enough to implement using many of the publicly available user interface toolkits or using the native Software Development Kits (SDK) that are available for modern desktop and mobile operating systems.

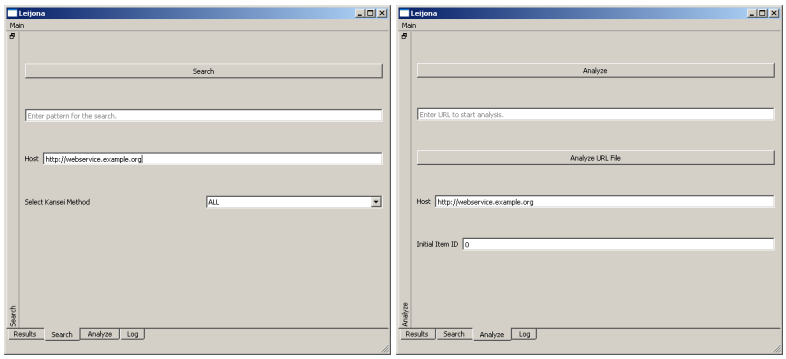

Figure 4. Test Application, search (left) and analysis (right) views.

Figure 4 shows the simple user interfaces for the video search (left) and analysis (right) functions. In the search view the keyword can be typed to initiate the search. The view also enables the user to choose whether he/she wants to target the search to *ALL* extracted keyword/value pairs or search only for the pairs extracted by a specific method (frequent, dominant or emergent). The selection of the method only affects the ranking of the results. All keyword/value pairs are always visible in the result list, as can be seen in the Figure 5. From the Analyze view (right side of Figure 4), the user can initiate the video analysis. This can be done either by submitting a single URL or by giving a list of URLs in a single file. The *Initial Item ID* is a unique identifier given for the video, and will be automatically incremented by the Test Application if the user provides the initial value in the case of multiple URLs. The value corresponds to the element of the same name visible in the example, in subsection 2.2.1.

In the result list (Figure 5) the results are generally listed in descending order with the best matching content on top. An exception to this can occur if the user does not target his searches to a specific Kansei method (see Figure 4) because the results (keyword/value) pairs are not necessarily comparable to each other, but they are always ranked by the keyword values. For this reason, one should be careful when observing the results. From the result list the user can also select the video name to view that particular video using the default video player application installed in the operating system. This way, the user can easily play the videos in the result list, and compare whether the videos match the keyword or not. The Figure 5 also shows the log view. The log view shows the time taken to execute the operations.

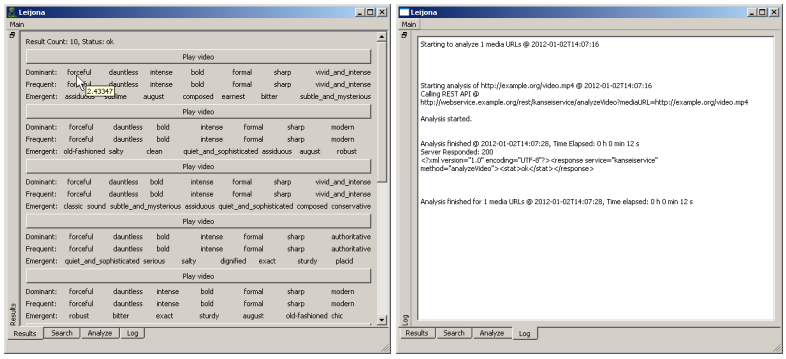

Figure 5. Test Application, result (left) and log (right) views.

2.4. Video Content

For a meaningful user studies, the content should be as close to possible to the content that the user might actually be watching. Fortunately, there are many sources available in the Internet that can be used for content acquisition. Unfortunately, often these sources do not come without problems. One of the main issues with online content is that the actual media (source, file) is often hidden and must be resolved using case-specific crawlers. This is the case with many of the popular video sites, such as YouTube [26], MetaCafe [17] or Yahoo [25]. In a simple case the download link must be parsed from a HTML page, in the worst case the video is only available by using a browser plug-in (such as Adobe Flash Player [1]). In addition, the online content can become unavailable at any time, or the direct download link for the video file can become invalid and need to be resolved again. As the research work was done in co-operation between Japanese and Finnish universities the region and country restrictions enforced by online content providers also caused problems – some of the content was available from Finland, but not from Japan, and vice versa. These issues made direct, repetitive and consistent use of online content problematic.

To overcome these problems, but still keeping in mind the requirement for real-life content, it was decided to pre-download the video files used for the testing and have them hosted on a dedicated server. In addition to guaranteeing the availability of the files, this also solved the possible issues that could be caused by the varying network bandwidths available from the online video services. The pre-downloaded dataset consisted of a total of more than 170 videos downloaded from well-known sources. When downloading online content, it should be noted that in the case of online video content providers [27], downloading may be forbidded by the Terms of Service rules. In some jurisdictions, even private, educational and research use of copyrighted material might be prohibited, which can make it harder to use real-life video content. The dataset collected for our testing purposes consisted of the following videos:

- The newest 50 Anime Videos uploaded to YouTube [26] on September 5th, 2011.
- A collection of 50 movie trailers from Aol. [2], Yahoo [25] and Warner Bros. [23].

- The 50 most watched music videos from YouTube [26] (week of August 30th, 2011).
- Various high definition videos from Xiph.org [24].
- Around 20 test videos were captured using a smartphone.

The quality settings for anime and music videos were: resolution of 360p and approximate bit rate of 600 kbit/s. The anime videos were encoded with MP4/h.264 and the music videos were encoded with FlashVideo/h.264. The movie trailers had a resolution of 480p with approximate bit rates of 2500kbit/s encoded with QuickTime/h.264. Anime videos, movie trailers and music videos were downloaded with the *normal quality* offered by the online content providers. After resolving the direct links and downloading the videos, it was observed that some of the videos (namely music videos) contained only a single image throughout the entire video clip. For a method that only analyzes the picture, this is not a very good starting point. Nevertheless, from the user's point of view the feeling or mood of the video is created even with videos with only a single frame. Also, there were huge variations in video lengths, with durations ranging from a few seconds to nearly half an hour. No videos were excluded from the dataset on the basis of the content, but it is worth bearing in mind these differences when running the test cases. Due to the limitations of availability, three music videos were not downloaded – two of the videos had country restrictions concerning Japan, and one video was marked as improper by the content provider.

The smartphone videos were captured using a Nokia C7-00 smartphone with high definition settings for 720p videos. The videos were of variable length (from 10 seconds to 45 minutes). The purpose of the self-captured videos were to allow both the testing of user-generated content, and also to make sanity checks for the analysis algorithms – by capturing videos ranging from a blank wall (still scenery) to random fractal shapes (movement and abstract shapes).

3. Tests Cases

The tests have two parts: testing the computing performance and testing the usability of the search results. The first part is fairly straightforward, consisting of running the collected dataset through the video analysis and measuring the time. Depending on the results the Video Analysis Back-end may need optimizations – a better selection algorithm for the video frames to be analyzed is one option. Based on the preliminary tests the time spent on relaying the analysis request through the Web Service is negligible (a few hundred milliseconds) compared to the time required to analyze a video file (from 5 to 30 seconds depending on the video file length and bit rate). It should also be noted that as the video analysis is based on color, the video content should be pre-processed before analysis. Letterbox formatted videos can offset the analysis by showing dark areas which should not be included in the analysis. Additional problems can be created by using user-generated, poor quality content with blurry or shaky video and possible color distortions.

The second part of the test process is more difficult. In our case a color impression matrix with 182 words (adjectives) representing human emotions was used. The association of colors with emotions varies from one person to another and also between different cultures. Because of this the color matrix should be kept relatively simple and universal.

In practice, an universal matrix that is usable for every user could be very difficult to make, and the matrix may need to be user specific or case specific. In addition to the color-emotion associations, the selection of adjectives should be validated. In the optimal case a list of often-used adjectives representing human emotions should be collected and the color representation for these words should be created. The adjectives could – at least in theory – be collected from the web's search engines, but whether color impressions for the words could be created easily is unknown.

In our preliminary tests a generalized matrix was used to extract words for the videos of the dataset. After extraction, the videos were shown to a limited number of users (13 members of research staff from TUT Pori Unit), who rated (with a score of 1 to 5) how well the keywords matched the video content in their opinion. Based on these tests, the adjectives chosen for the test were the most common cause of complaints by the participants. Many of the words were thought to be difficult to understand, which was partly due to the fact that the words were English, but none of the participants were native English speakers. In other cases participants disagreed about what a specific word should mean. For example, some people felt that the use of the keyword *forceful* required the video to include aggressive or intense action, but others felt that even a video with a little movement could be considered forceful if the singer's voice in a music video sounded forceful. In this case the weakness of only processing the picture became clear; in real-life applications – especially in the case of music videos – the audio track should also be analyzed. Another example could be the word *metallic*, which could mean that the video has elements related to robots, machines or technology, or that only the colors of the video were metallic in appearance (for example, like silvery).

In the preliminary tests, differences in people's watching habits were also observed. Some participants only glimpsed parts of the video, while others watched the entire video clip before making their assessment about their feelings. Repetitive watching of – not necessarily interesting – video content can also be tiresome, which could affect the user's impression of the video.

Based on the preliminary tests, the results were somewhat mixed. In the case of simple keywords, such as *cute*, *simple* and *forceful*, the results were generally good, and the participants felt that the extracted keywords matched the video content. But when more complex words such as *sharp*, *formal* or *dauntless* were included in the tests, the participants felt that the matches were not very good. In addition, with more complex words the variance in the answers grew, with some participants rating the matches as excellent, but others rating the matches as poor. Based on these observations, more care should be put into selecting of the keywords.

4. Conclusions and Future Research

The goal of the research was to design a test environment that could be used for Kansei-based video analysis using the Mathematical Model of Meaning. The system implementation can be considered to have met the assigned goals. The tests run with the system were only preliminary and user studies targeted for a much wider audience should be performed to allow a more conclusive and definite analysis of the usability of the video analysis. Still, some observations can be made. The creation of a good color impression matrix can be difficult, and in practical implementations the matrixes may have to be

personalized for the user. The personalization of the matrixes was not implemented in this test environment, and it is a possible direction for future research. In real-life implementations, it could be beneficial to combine multiple methods of analysis. Different methods could be used to complement each other, such as running both Kansei-based analysis and shape recognition for the picture, and tonality analysis for the sound.

While the analysis of computing performance can be quite simple, the other side of the study – observing human emotions – can be very challenging. There is no easy way to assess the usability of Kansei-based analysis. The results (keywords) created by the analysis are meant to relate to human emotions, and the only way of to validate them is to perform additional user studies (that is, running videos through the analyzer and asking users for feedback). The field of (color) psychology may provide a starting point for studies, but whether human emotions can fully be transformed into parameters and usable metadata remains to be seen.

Acknowledgements

This work is partially funded by the Finnish Funding Agency for Technology and Innovation (Tekes). The MediaFusion platform used in the test environment was provided by PacketVideo Corporation. The research was made possible by the good and long-lasting co-operation between TUT Pori Unit and Keio University SFC, and the commonly organized researcher exchange in the summer of 2011.

References

[1] Adobe Flash Player, http://adobe.com/flashplayer, retrieved January 2nd, 2012.
[2] Aol. Moviefone, Aol. movie trailers, http://www.moviefone.com/, retrieved January 2nd, 2012.
[3] Chilingaryan, S.: The XMLBench Project: Comparison of Fast, Multi-platform XML Libraries, Database Systems for Advanced Applications, Lecture Notes in Computer Science, (2009), Volume 5667/2009, pp. 21-34.
[4] The FFmpeg project, http://ffmpeg.org, retrieved January 2nd, 2012.
[5] IETF Network Working Group: RFC 1321, The MD5 Message-Digest Algorithm, http://tools.ietf.org/html/rfc1321, (1992).
[6] Imai, S., Kurabayashi, S., Ijichi, A. and Kiyoki, Y.: A music database system with content analysis and visualization mechanisms, in proceedings of Parallel and Distributed Computing and Systems (2008), Orlando, Florida.
[7] Kitagawa, T. and Kiyoki, Y.: Fundamental Framework for Media Data Retrieval Systems using Media-lexico Transformation Operator - in the Case of Musical MIDI Data, Information Modelling and knowledge bases XII, IOS Press, (2001), pp. 316-326.
[8] Kiyoki, Y., Chen, X. and Kitagawa, T.: A Semantic Associative Search Method for WWW Information Resources, WISE, (2000), pp. 230-237.
[9] Kiyoki, Y., Kitagawa, T. and Hayama, T.: A metadatabase system for semantic image search by a mathematical model of meaning, ACM SIGMOD Record, vol. 23, no. 4, (1994), pp. 34- 41.
[10] Kiyoki, Y., Kitagawa, T., and Kurata, K.: An adaptive learning mechanism for semantic associative search in databases and knowledge bases, Information modeling and knowledge bases VIII, IOS Press, (1998), pp. 345-360.
[11] Kobayashi, S.: Color Image Scale, Oxford University Press, USA/Kodansha International, Japan, (1992), ISBN 4-7700-1564-X.
[12] Kurabayashi, S. and Kiyoki, Y.: MediaMatrix: A Video Stream Retrieval System with Mechanisms for Mining Contexts of Query Examples, Database Systems for Advanced Applications, Lecture Notes in Computer Science, Volume 5982/2010, (2010), pp. 452-455.

[13] Kurabayashi, S., Ueno, T. and Kiyoki, Y.: A Context-Based Whole Video Retrieval System with Dynamic Video Stream Analysis Mechanisms, ISM '09 Proceedings of the 2009 11th IEEE International Symposium on Multimedia, (2009).
[14] Nagamachi M. (ed.): Kansei/Affective Engineering, CRC Press, (2010), ISBN 978-1-4398-2133-6.
[15] MediaFusion platform, http://www.pv.com/products/mediafusion, retrieved January 2nd, 2012.
[16] MediaMatrix, http://web.sfc.keio.ac.jp/ kurabaya/mediamatrix.html, retrieved January 2nd, 2012.
[17] metacafe the Video Entertainment Engine, http://www.metacafe.com/, retrieved January 2nd, 2012.
[18] Miura, N., Kurabayashi S. and Kiyoki Y.: An Automatic Extraction Method of Time- Series Impression-Metadata for Color Information of Video Streams, ICDEW '05 Proceedings of the 21st International Conference on Data Engineering Workshops, (2005), IEEE Computer Society Washington, DC, USA, ISBN 0-7695-2657-8.
[19] Nurseitov, N., Paulson, M., Reynolds, R. and Izurieta, C.: Comparison of JSON and XML Data Interchange Formats: A Case Study, in proceedings of The International Conference on Computer Applications in Industry and Engineering (CAINE), (2009), San Francisco, California, USA.
[20] Peng, D., Cao, L. and Xu, W.: Using JSON for Data Exchanging in Web Service Applications, Journal of Computational Information Systems 7: 16 (2011), pp. 5883-5890.
[21] Qt cross-platform application and UI framework, http://qt.nokia.com/, retrieved January 2nd, 2012.
[22] Sasa, A., Krisper, M., Kiyoki, Y., Kurabayashi, S. and Chen, X.: A Personalized Recommender System Model Using Colour-impression-based Image Retrieval and Ranking Method, ICIW 2011 proceedings of the Sixth International Conference on Internet and Web Applications and Services, (2011), St. Maarten, The Netherlands Antilles.
[23] Warner Bros. movie trailers http://www.warnerbros.com/, retrieved January 2nd, 2012.
[24] Xiph.org: Test Media, a repository for freely redistributable test sequences. Retrieved December 29th, 2011.
[25] Yahoo! Movies, http://movies.yahoo.com/trailers, retrieved January 2nd, 2012.
[26] YouTube, http://www.youtube.com, retrieved January 2nd, 2012.
[27] YouTube, Terms of Service, http://www.youtube.com/t/terms, retrieved January 2nd, 2012.

Publication IV

Rantanen P., Sillberg, P. and Soini, J., "Content Analysis System for Images", in Proceedings of the 16th International Multiconference Information Society, IS 2013, Volume A, 7-11, Ljubljana, Slovenia. Josef Stefan Institute, pp. 241-244, October 7-11, 2013.

Original publication is available at `http://is.ijs.si/is2013/zborniki.asp?lang=eng`.

CONTENT ANALYSIS SYSTEM FOR IMAGES

Petri Rantanen, Pekka Sillberg, Jari Soini
Department of Software Engineering
Tampere University of Technology - Pori
P.O.Box 300, FIN-28101 Pori, Finland
Tel: +358 40 8262890; fax: +358 2 6272727
e-mail: jari.o.soini@tut.fi

ABSTRACT

This paper presents a concept architecture developed for managing digital data. The paper introduces the basics of a simple task-based system, which can be used for scheduling content analysis tasks on remote back-ends. The interaction between client software, front-end service, and content analysis back-ends are illustrated in combination with an optional feedback mechanism designed for improvement of analysis results. The paper describes the system components and overall system architecture, and also explains how task generation and data analysis are implemented. The presented concept architecture is applicable to various forms of media, but the focus of this paper is on image content.

1 INTRODUCTION

The research topic deals with the challenges of managing vast amounts of data. Currently, there is a huge amount of digital data stored in many different kinds of digital storage, even at individual level. Moreover, the stored data is not only big, but it is also unstructured. The problem is simply how to manage all this stored digital data. Tampere University of Technology is involved in a large national research project called Data to Intelligence (D2I). The aim of the D2I project is to study and develop methods and tools for the management, processing, and utilization of large amounts of data captured from the environment, Internet and many other sources. In addition, new business opportunities are being developed around this material. As part of this project, the study focuses on examining the challenges related to processing digital data automatically. For example, humans can process text, images, videos, audio, and other forms of data effortlessly, but these are the most difficult to process automatically by computer.

2 TASK-BASED ANALYSIS SYSTEM FOR IMAGE CONTENT

The system consists of a front-end service, which provides a REST API for the clients, and can be used to execute search queries, modify content on the front-end, and can also be used to connect external accounts of 3rd party content storage services. The front-end does not store any actual content, but only maintains a metadata base, which is used for resolving the links to the content stored on external accounts. In addition to being capable of retrieving content and the associated metadata located on external accounts, the front-end can utilize various content analysis engines for the process of extracting additional metadata. In the scope of this paper, the focus is on content analysis engines capable of image analysis, and more specifically, content-based image retrieval [1, 2].

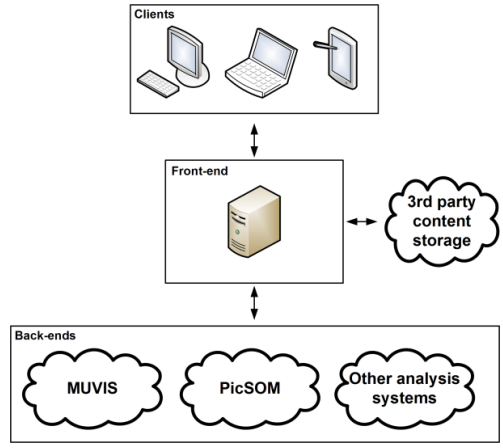

Figure 1: *Overall system architecture.*

The main purpose of the multiple analysis back-end (content analysis engine) approach is to make it possible to gather as wide a range of metadata as possible – as generally most systems are more suitable for one task, and do not perform as well with other types of tasks. For example, a back-end capable of feature extraction cannot do description text summarization. In principle, the multiple analysis back-end approach could also be used for load-balancing if there are multiple mirrors of the same back-end available, but this would require knowledge of the current load status of said back-ends, or the back-ends should be dedicated to the tasks provided by the front-end. For these reasons, in our scenario, the back-ends are responsible for their own load-

balancing and resource allocation. In an ideal scenario, all analysis results would be shown to the user immediately after the content has been uploaded, and with a relatively small content base this can be achieved, but in the case of thousands, or tens of thousands of images, there may be a significant delay before the analysis operation is completed. Depending on the overall quality of the results, this may force the user to recheck his/her images after the analysis has finished. The total time required for analysis depends on the content and is difficult to estimate precisely. In practice, the results are seldom perfect, and thus, a mechanism for user feedback is required. The feedback may be requested during the initial upload of the content, or some time after, depending on the desired use case. The overall system architecture is shown in Figure 1.

Google's Picasa was our choice for the content storage service, but any of the alternative services (Flickr, Amazon, etc.) could have been chosen as well, as the only requirements were the ability to store image content and a public REST API, which could be easily used to access the image details from the user's account. There are several advantages in using a public service. It saves the trouble of implementing "just another content storage," and also enables the use of network and storage capabilities not necessarily otherwise available for research purposes. In addition, this approach makes the testing process easier, as the participants can use the content provider's APIs and well-known web clients for adding, updating, and removing content. If the test cases are very complex, it may be necessary to implement an additional client for browsing the content, or for executing other tasks that are not possible with the default client of the content provider. Whether this kind of split between two (or more) client applications is an issue or not depends on the use cases, but for us, the resources and time saved by not implementing new image storage were well worth it.

In the current implementation, two analysis back-ends, MUVIS [10] and PicSOM [11, 12, 13] are used, which have been provided by the partners of our research project. If any other back-end had been used, that back-end would naturally have implemented our interface specifications, which is not usually possible with commercial systems. Unfortunately, no commonly used generic and free protocols exist for task scheduling to external systems. Also, there is an apparent lack of open public services that could be used for advanced media analysis (such as feature extraction), which are not limited to the extraction of basic media details (e.g. Exchangeable Image File Format metadata [14]) that can easily be extracted without the use of external analysis back-ends. It should be noted that some metadata could be extracted by uploading the content to one or several of the publicly available content storage systems, and then retrieving the metadata generated by those systems, but in addition to being a somewhat cumbersome process, it may also be in violation of the terms of service agreements of the services in question – there may be limits on what purposes the services can be used for, and what it is permitted to do with the automatically generated content.

The basic concept of the system is based on the work done on previous research projects [3, 4]. An important point to make is that even though the system is suited to work as an end-user system, its primary purpose is to work as a test platform for content analysis engines in the scope of the D2I project. From this point of view, the client interface is secondary, and the communication protocols for the back-ends are of greater concern. The communication with the back-ends is basically two-fold. In the first phase, the back-ends are provided with a workload, which they should analyze, and report their findings back to the front end. In the second phase, these results should be validated by the user, either by asking for direct feedback [5, 6] or indirectly by automatically generating feedback based on the user's actions [7, 8, 9]. These two phases form the two basic task types of our system: the *analysis* task and the *feedback* task.

3 TASK-BASED APPROACH

The communication between the front-end and back-ends is achieved by using tasks and task responses. The tasks contain information about the type of the task and the actual contents (workload). The task workload generally consists of a list of images, in which each image contains the required base details (e.g. image URL, identifier), and any optional metadata (e.g. the description of the image) that might be useful for the back-end, but is not strictly speaking required for the completion of the task. The "usefulness" of metadata is decided by using a list of predefined data group classifiers, which are specific to a certain type of a task and the back-end in question – i.e. the task contents may vary from back-end to back-end, and from task type to task type, if needed. An example of a data group classifier could be "keywords," which, if present in the configuration, would trigger the inclusion of keywords associated with images into the generated task.

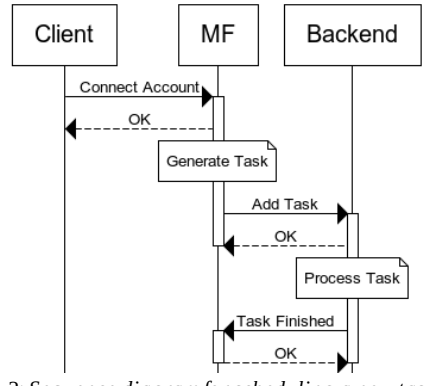

Figure 2: *Sequence diagram for scheduling a new task on an analysis back-end.*

For the content analysis, the current implementation of the system provides two task types: *analysis* and *feedback*. The *analysis* task is used for requesting media analysis work from a back-end, such as tag or keyword extraction. A simple use case illustrating the task of type *analysis* is provided in Figure 2.

In Figure 2, the analysis task is generated after the user has connected his/her account to the system. The task generation may also be triggered by a web crawler, or by periodical synchronization with a previously connected account. If the account contains valid image content, the metadata of this image is stored in a database on the front-end, and based on this data a new task is generated for submission to back-ends. The task procession is always asynchronous, and the analysis back-end is free to initiate the actual analysis whenever it is most convenient. After the task has finished on the back-end, a notification is sent to the front-end. This notification message should contain any extracted metadata, and any possible error situations that occurred during analysis. Depending on the severity of the error, the image in question may be included in a new task, scheduled for a later execution – corrupted image data will never be successfully analyzed, but a temporary network error may resolve itself after a while. It should be noted that the task finished notification may contain only partial analysis results, which means that the back-end may provide the results progressively, or try the analysis again at a later date if needed.

Upon generation of a task, the base details of the task are added to the front-end database. These details include, for example, the list of image identifiers contained in the task, the ID of the task, the back-ends which should partake in the execution of the task, and optional user details. The back-ends may provide results for the tasks as long as the tasks are available. The tasks may become unavailable after a certain time period (expired tasks), or if all the content for the task is deleted by the user.

The user details can be passed to the back-ends, or the task can behave entirely in an anonymous manner, in which case the back-ends will not have any details about who owns the image content. The user details will never contain sensitive data about the user (such as a user name or password to the external account), and will only contain the user identifier assigned by the front-end, which can be used by the back-end to differentiate the users from each other. The URL links to image content are always provided as links pointing to the front-end, which will handle authentication to the external account, and provide necessary URL redirection for reaching the image content. This functionality can be used to hide the real user credentials from back-ends, but it also makes the back-end implementation simpler, because they only need to implement the interface for communicating with the front-end, and not all of the various authentication schemes used by the supported services. The disadvantage of this approach is the increased traffic on the front-end generated by the URL redirection requests.

The list of images and their details are dynamically generated based on the image identifiers and the available metadata. For the basic use case, depicted in Figure 2, this approach has no practical advantage, but using the system also makes it possible to give the back-ends only the task ID when scheduling the task. Thus, the back-ends may retrieve the task details just before starting the analysis, and in this way receive the most up-to-date metadata, which may contain results from previously finished analysis runs (i.e. from other back-ends). The back-end may always disregard the details provided in the *Add Task* call, and manually retrieve the task details. Disregarding the details may come into question if the analysis on the back-end is for some reason started after a very long time has passed since the *Add Task* call was received. For the basic analysis functionality this may not be required, as new images are never added to existing tasks, and submitting results for images that have been removed from the task is not considered an error. If the addition of new images to the old tasks were allowed, this would require more complex synchronization logic between the front-end and the back-ends. The same functionality can be achieved in a simpler way by creating new tasks when adding new content.

3.1 Task Results

When submitting the result (*Task Finished* call, in Figure 2), the back-ends cannot directly modify the details of the images. Instead, each image contains an *object list*, which may contain objects created either by the user, the front-end, or by the back-ends. These objects contain predefined fields for the object name, value, object type, and any special fields associated with a specific object type. The currently supported types are *metadata*, *keyword*, *face*, and *object*. *Metadata* is the simplest type, containing a basic name-value pair of information, such as GPS coordinates. *Keyword* (or "tag") and *face* are special types of metadata, and they are used when keyword-based queries or people-based queries are performed by the user. These two types may also contain coordinate (or area) information, which can be used to pinpoint the data to a specific position over the image. The face recognition data is in our case extracted by one of our own back-ends, but the metadata could also be extracted directly from the external account if the public API of that particular service supports the functionality. *Object* type is a generic container, which is stored as-is on the database. In addition to the type-specific fields each object contains an *object ID*, which should be provided by the object creator, such as the user or the back-end. If the ID is missing, it will be generated automatically – internally, a separate ID is also generated for each object, to preserve ID uniqueness. A single object can be associated with multiple images, in which case modifications to a single image can be reflected to other images. Upon submitting a new object, the

submitter is also recorded on the database, and by using the submitter ID (user identifier, or back-end identifier) and the *object ID*, it is possible to update pre-existing objects. This enables the user to update his/her own objects, and the back-ends to update their earlier results. Any object creator can modify only objects it has created itself, with some exceptions; the user can always modify objects that are associated with his/her content whether they were originally created by him/her or the automated extraction process; and back-ends can never overwrite object information explicitly modified by a user whether the object was originally created by the back-end or not. There is a possibility that a back-end might later on provide results that would be better than those the user had originally accepted, but allowing the back-end to override the user's modifications generates various usability issues. The "correctness" of a result can be strongly related to a particular user, and making an automatic foolproof estimation of improvement can be very difficult. It could be argued that the user would be more willing to preserve his/her own modifications, even if they are – relatively speaking – worse, as opposed to the idea of an automatic system repeatedly updating his/her content. If the updates happen *very* rarely, they could be confirmed manually by the user.

3.2 Feedback Tasks

The *feedback* task is often "indirectly" created by the user. Certain operations, such as changing the image description, or updating a pre-existing object can trigger the creation of a feedback task. The purpose of the *feedback* task is both to provide the back-ends with statistical information about how the users use their objects, and to enable self-learning functionality in the back-ends. If the back-ends cache information about previously analyzed images and the objects they have created, the feedback information can, in principle, be used to enhance future analysis results or point out mistakes in previous results. As the *Task Finished* may be called at any time in the future as long as the task is valid, the back-end may update its previous analysis results based on the feedback. At the very least, the information should help the back-ends to discover results that are repeatedly incorrect, or do not generally match the users' perception. If the *feedback* task also contains user details, the feedback can be targeted to a specific user, and it could affect the future analysis results of that particular user.

4 SUMMARY AND FUTURE WORK

This paper briefly presented the basic operating principles of a task-based content analysis system. The system is one solution for improving the management of a huge amount of digital data, in this case, images. The full documentation for the system, including the proposed protocol specification, is to be published as the research project (D2I) progresses. In addition to improving the basic system, research is ongoing into methods for collecting user feedback. For long-term research, there are also plans for extending the system to implement audio and video capabilities.

References

[1] M.S. Lew, N. Sebe, C. Djeraba, R. Jain. Content-Based Multimedia Information Retrieval: State of the Art and Challenges. ACM Transactions on Multimedia Computing, Communications and Applications. Vol. 2, No. 1. pp. 1-19. 2006.

[2] R. Datta, J. Li, J.Z. Wang. Content-Based Image Retrieval – Approaches and Trends of the New Age. MIR'05 Proceedings of the 7th ACM SIGMM International workshop on Multimedia Information Retrieval. pp. 253-262. Singapore. 2005.

[3] P. Rantanen, P. Sillberg, H. Jaakkola, T. Nakanishi. An Asynchronous Message-based Knowledge Communication in a Ubiquitous Environment. Lecture Notes in Computer Science Vol. 6193. pp. 434-444. 2010.

[4] P. Sillberg, S. Kurabayashi, P. Rantanen, N. Yoshida. A Model of Evaluation: Computational Performance and Usability Benchmarks on Video Stream Context Analysis. Information Modelling and Knowledge Bases XXIV. Vol. 251. pp. 188-200. 2013.

[5] N. Vasconcelos, A. Lippman. Learning from user feedback in image retrieval systems. Advances in Neural Information Processing Systems. Vol. 12. pp. 977-986. 2000.

[6] Y. Rui, T.S. Huang, M. Ortega, S. Mehrotra. Relevance Feedback: A Power Tool for Interactive Content-Based Image Retrieval. IEEE Transactions on Circuits and Systems for Video Technology. Vol. 8, No. 5. pp. 644-655. 1998.

[7] P. Maes, R. Kozierok. Learning Interface Agents. AAAI-93 Proceedings of the 11th National Conference on Artificial Intelligence. Washington D.C., USA. 1993.

[8] E. Agichtein, E. Brill, S. Dumais. Improving Web Search Ranking by Incorporating User Behavior Information. SIGIR'06 Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 19-26. Seattle, Washington, USA. 2006.

[9] L.A. Granka, T. Joachims, G. Gay. Eye-Tracking Analysis of User Behavior in WWW Search. SIGIR'04 Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 478-479. Sheffield, South Yorkshire, UK. 2004.

[10] MUVIS, A System for Content-Based Indexing and Retrieval in Multimedia Databases. http://muvis.cs.tut.fi/. 2013.

[11] J. Laaksonen, M. Koskela, S. Laakso, E. Oja. PicSOM – Content-based image retrieval with self-organizing maps. Pattern Recognition Letters Vol. 21. pp. 1199–1207. 2000.

[12] J. Laaksonen, M. Koskela, S. Laakso, E. Oja. Self-organizing maps as a relevance feedback technique in

content-based image retrieval. Pattern Analysis & Applications. Vol. 4. pp. 140–152. 2001.
[13] The Content-Based Image and Information Retrieval Group, Department of Information and Computer Science, Aalto University. http://research.ics.aalto.fi/cbir/. 2013.
[14] Exchangeable image file format for digital still cameras: Exif Version 2.3. http://www.cipa.jp/english/hyoujunka/kikaku/pdf/DC-008-2010_E.pdf. 2010.

Publication V

Rantanen P. and Sillberg, P., "Event Calendar for Internet Data Sources", in Proceedings of the 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, Croatian Society for Information and Communication Technology, DOI: 10.1109/MIPRO.2014.6859721, pp. 1035-1040, May 26-30, 2014.

Event Calendar for Internet Data Sources

P. Rantanen* and P. Sillberg*
* Tampere University of Technology/Pori Unit, Pori, Finland
{petri.rantanen,pekka.sillberg}@tut.fi

Abstract – Several nations in Europe have released formerly closed data sources collected by government agencies and municipalities. These open data initiatives have enabled new and innovative services. The research presented in this paper is funded by the European Union Regional Fund as part of the Satakunta Regional Centre for Economic Development, Traffic and Environment funding program, and by the City of Pori. One of the goals of the research is to study how open data can be utilized technically. This paper presents technologies found useful for the implementation of a service utilizing open data. These technologies are presented in this paper as part of an event calendar service that uses open data sources available in the region of Satakunta, Finland. The overall architecture of the service is described briefly, and the paper focuses on the open source components used to implement the user interface, front-end service, and database. Despite the use of region-specific data sources, the utilized solutions i.e. hybrid databases, search platforms, and HTML5 technologies can be exploited in the design of other services.

I. INTRODUCTION

The research presented in this paper is part of the AVARAS project. AVARAS is an Open Data Project in the Satakunta Region in Western Finland conducted by Tampere University of Technology, Pori Unit. The idea of it is to study and implement business opportunities related to Open Data. It is a regional activity with close connections to more than ten SME-sized software companies that are used to guide and direct the project activities. The project is funded by the European Union Regional Fund as part of the Satakunta Regional Centre for Economic Development, Traffic and Environment funding program. [1]

One of the main goals of the project is to study how open data sources can be utilized technically. The research is conducted using pilot applications, which will be openly available for public use. Their purpose is especially to publish interfacing technologies and interface specifications, which can be re-used in new applications. Similar activities are taking place in other regions in Finland [2], [3]. This paper presents one of the pilot applications designed in the AVARAS project.

The application presented in this paper is an event calendar service, which utilizes event data available on the public Internet. The service does not require any input from the event publishers, and can be set to crawl any number of sources automatically. There are many event calendars available on the Internet and the idea of an event calendar is in no way a new one, even though one could argue that there are not that many calendars that can list external events without some kind of input from the event publisher or event provider. Nevertheless, the main reason for choosing an event calendar application in particular was the type of container it provides. Basically any data can be thought of as "an event": the daily weather forecast, road accident reports, sports events, or movie showtimes – which are used as an example case in Section V of this paper. In a way, the idea of an event calendar can be generalized to contain almost any kind data. This paper does not go into too much detail when describing the basic calendar functionalities, but attempts to illustrate how the chosen technologies could be used both in this use case and perhaps in a more general way outside the event calendar context by describing the advantages and disadvantages of the chosen technological solutions.

As mentioned above, the example shown in Section V uses details of movies from the local cinema as a data source, though any other data source could have been chosen as well. The other sections of this paper are as follows: Section II gives a short overall presentation of the chosen system architecture; Section III describes the chosen database solution; Section IV explains the designed REST interface; Section VI discusses some of the issues related to the application; and finally, Section VII summarizes the paper.

II. SYSTEM OVERVIEW

The system introduced in this paper is an effort to utilize publicly available data in the region of Satakunta, Finland, and also test the usability of open source software components and libraries. The selection was made partly by suggestions from the steering committee of the project, and partly guided by academic ambitions. Fig. 1 illustrates the basic organization of the system. There are three distinct components: data storage, front-end service, and client, which are described in more detail in Sections III, IV and V, respectively.

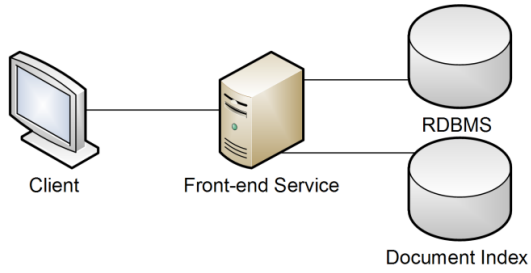

Figure 1. System overview

The *Client* is in principle an HTML web page, which accesses the *Front-end Service's* RESTful (e.g. [4], [5]) interface to retrieve data, and then visualizes this data in a calendar view. The client allows the user to narrow down the dataset by entering filtering terms such as date, time, keywords and also free text searches.

The *Front-end Service* is an access point to the features of the system. It provides user authentication, RESTful API for data in XML format, and adapts the data sources to make them suitable for indexing and storing. The service is written in Java and the main open source components it utilizes are Apache Tomcat [6] and Spring Framework [7].

Our database solution includes features from both the traditional RDBMS and NoSQL [8] approaches. The *Document Index* uses the Apache Solr [9] search platform as it is specialized in indexing and searching, while the *RDBMS* is based on MySQL [10] and is used to store the document relationships and basic structural information such as user credentials.

III. DATABASE

The searching and indexing of data provided by open data sources is the key point of the system. The database solution must have good search capabilities, but it also must be able to handle loosely structured data. Different data sources are often provided in various formats, thus it may be difficult to parse the same data from them all. For example, one source could simply list the names of the performers while another source may contain performer names, email addresses, and groups they belong to.

The SQL approach with its well-defined table-column structure was considered too rigid for the data we might encounter. Of course, we could define all possible columns and leave them null if not available, but that would not be very practical. From the NoSQL approach we found a solution, Apache Solr, which had the capability to index and store the data, as well as sophisticated text search features.

The most interesting feature of Solr is its ability to index any kind of text in various ways. With relatively small changes and additions to the Solr core's basic setup, it is possible to enable request handlers such as full text search, "more like this," and auto completion [11]. As a security measure, the database section in our system is behind the front-end service. This means that the client cannot access features of the database directly. A simple call through API was implemented on the front-end service to handle requests.

There is often a need for storing relationships between documents, and implementing this feature was cumbersome with Solr alone. For storing simple data types, such as strings, texts, dates, or a list of strings, Solr can manage with no problems. Issues arise if there is a need to store complex and denormalized data within the same document. For example, take the situation where there is an event involving many performers in it, and for every performer we need to store a name, email address and URL for the website. There are several possible approaches on how to solve the issue:

1. Store performer data in synchronized lists inside the same event document. This requires some extra effort in marshalling objects, but Solr can still index the document effectively. Maintenance of the document or parts of it suffers from the denormalized data format and from the way Solr handles updates (complete overwrite, i.e. no partial updates).
2. Store performer data in another Solr core as separate documents, and store the relationship (i.e. performer ID) in the original event document. This approach allows easier maintenance and is closer to the traditional relational approach. Solr cannot handle joins as fluently as SQL does, so extra lookups and queries are required. Extra work is required for fully utilizing search and indexing in the Solr multi-core setup.
3. Store performer data in another Solr core as separate documents, but store the relationship (i.e. performer ID and event ID) in another database (e.g. MySQL). If the only task of this other database is mapping of relationships, a graph database (e.g. [12], [13]) might be an alternative choice for a SQL database. This hybrid – or polyglot [8] – approach is much like the previous case, but the handling of relationships might be easier. This increases complexity due to the use of another database technology.
4. Store both the event data and the performer data in the same Solr core, but store the relationship (i.e. performer ID and event ID) in a separate database. Use of a single Solr core simplifies indexing and searching. Storing of multiple types of documents in a Solr core can be achieved as Solr allows dynamic field assignments. The type of document must be known when the data is marshaled. Compared to the first case, maintenance of these smaller documents may be easier.

A combination of multiple databases is called the polyglot persistence model by reference [8]. Together they may create an ecosystem that is more powerful than the sum of its parts [8]. In theory, one could choose any combination of database techniques if using them together generates more benefits than drawbacks (e.g. code complexity, time to develop).

We chose the hybrid approach (3) for the event calendar implementation. In our use case the main event details and minor event details are stored in separate Solr cores, and the relationships are stored in a MySQL database. The minor event details can contain information such as event location and event organizer, or sub events which belong to the main event. In principle the minor event details could be stored directly to relational database, but this would prevent using Solr's text indexing and search functionality. On the other hand, the details could be merged with the main event details, and stored in a single core as described in the approach 1 above. In practice this would require flattening of very complex objects to key-value pair lists, which is more difficult than separating the less important data into individual cores.

Naturally, dividing the data between multiple cores will have an effect on the complexity of search queries and the search performance if the queries need to be targeted to multiple cores. In addition to a performance gain, in our use case there is also an additional advantage achieved by flattening certain important data with the main event details, which is perhaps not immediately evident. Event details often contain various performer details, such as description of a band, movie studio or names of the artists. These descriptions generally vary depending on the data source and analyzing the details to form a single combined entity is difficult. A simple solution to this problems is to ignore the performer-event relations altogether and flatten the data into a single core. This will cause all details to be duplicated and appear slightly different for each event, but it will also enable the use of Solr's text search features when querying for the event details. From the text indexer's point of view these minor differences in the descriptions are irrelevant.

IV. Front-end Service

The front-end service implements interfaces for three kinds of functionality: user management, suggestions, and search. In our demo implementation the user management is limited to basic functionality, i.e. creating a new user and modifying user details (such as password and email address). User authentication is provided using the Spring Security [14] framework and in principle supports all features supported by the framework. The reason for implementing only a minimal set of user authentication and authorization features is the nature of the processed data, which consists of publicly available content. There is no need to check user authorization when accessing the content. Of course, the service could be extended with the possibility to add user-created content, but this would complicate the implementation and is not the focus of our current demo implementation. The user management could also be utilized for personalized calendars, but this functionality is not currently implemented.

The suggestion API provides the user with search suggestions. It is implemented using the text indexing features found on Solr and works similarly to the functionality found on many search engines. For example, if the user types "Hobb", based on the indexed content the system may suggest search terms "Hobby" or "Hobbit".

The search API provides the functionality required for content queries and offers features such as free-text search, content filtering, sorting, and paging. Many of the currently available open source databases directly offer interfaces for the aforesaid features [15], [9], which can be easily used over HTTP by using GET or POST. In this case content filtering refers to filtering by values (e.g. only return events taking place between January 1st and January 10th), though similar functionality could also be used to implement filtering for improper content (e.g. do not show adult content to anonymous users or to children).

Depending on the use case, these interfaces might be all that is necessary. The limitations of the APIs are not so much on the basic functionality, but more on the user management. The NoSQL database APIs often do not enforce – or even provide only a very limited – user authentication. In a way this is a shame, as in many cases the REST interface provided by the NoSQL databases could be all that is needed for the query operations. Then again, perhaps the idea is, that it is not the task of the NoSQL database to perform user authentication. Of course, in many cases this does not matter so much as it is often the case with web applications, that the user authentication is performed by the front-end service or by some kind of mediator interface, and the user does not directly interact with the database. The similar approach is possible with Solr and with any other NoSQL database. If a complex authentication scheme is required it might be better to implement a front-end service with interfaces combined with user authentication and authorization. The demo implementation presented in this paper uses the aforementioned design pattern by providing a separate REST interface for query operations, which in turn will delegate the requests to the database when appropriate.

Another advantage received from implementing a separate front-end service is the additional control gained over the requests. This enables simplifying the search queries or limiting the set of available search filters or even resolving the XML element (or JSON object) names given in a query to database field names to be used for sorting and filtering. In particular, mapping element names to database fields is often required when the actual field names defined in the database are hidden to end users or developers using the API. As document databases generally offer output in the form of XML or JSON, it is possible to use the field names defined in the document directly in the response returned to the user, but this is generally not a good practice – it is much easier to change the internal implementation when the fields defined in the database layer are not directly being used by the users of the service API. Also, defining separate field names and XML element names is often a minor task, for example by using annotations in Java. Both the Solr Java implementation (SolrJ) [16] and Spring Data framework [17] provide annotation support for defining Solr field names for Java objects, and the annotations can be combined with Java Architecture for XML Binding (JAXB) [18] or Google GSON [19] to provide XML and JSON responses. This way the response Java objects that are deserialized from Solr query responses can be directly serialized to XML or JSON for external service API responses.

On Solr, database queries cannot modify the database content and thus the risk of code injection can be mitigated by rejecting all other user requests except queries. In our use case, there is no risk of users retrieving content they are not authorized to view by running a malicious query, as all of the content is publicly available. Also, it should be noted that implementing user preferences does not necessarily require user authentication. For example, if the user always wants to see the latest events or only events of a certain type (such as sports), the preferences can be stored on the client software or in local storage [20] on a web browser and from these preferences the required filter parameters can be generated and appended to the queries.

V. CLIENT

Fig. 2 illustrates a calendar with events retrieved from XML API [21] provided by a local cinema in the City of Pori. The calendar view could include any events indexed by the front-end service, but for simplicity only movie events are shown in this example. The view in the figure is a screenshot from a web browser. The main page layout is created using the w2ui JavaScript UI Library [22], which consists of plugin extensions for the popular jQuery library [23] and the calendar view itself is an embedded component created with the FullCalendar jQuery plugin [24]. There are many open source alternatives that can be used to implement web user interfaces (e.g. [25], [26]) and calendars (e.g. [27], [28]), and it is possible to implement a new user interface from scratch using basic JavaScript or jQuery, but in many cases the pre-made plugins and libraries provide a better starting point. The choice between the various alternatives is partly use case-specific as not all libraries provide similar functionality, and partly based on the developer's preference or pre-knowledge of a particular library. In this use case we use movie details from the local cinema.

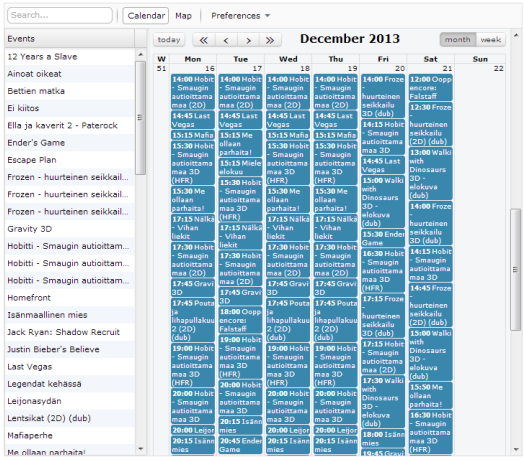

Figure 2. Calendar view

The calendar view in Fig. 2 shows all events currently listed in the events panel on the left. In the example case, no search terms have been given and thus all movie events known by the system are shown. The search field at the top left accepts both free text search and the basic query syntax of the Solr query parser. The Solr query syntax is based on the Lucene Query Parser syntax, which can be somewhat cryptic for end users. There are alternatives for the default parser such as the Extended DisMax Parser [21], which can provide better usability. However, in general, users will most likely prefer free text search over syntax-based queries, and it is better to create filter parameters and additional search terms programmatically and append them to the search query given by the user. The additional parameters can originate from, for example, checkbox selections or user preference options.

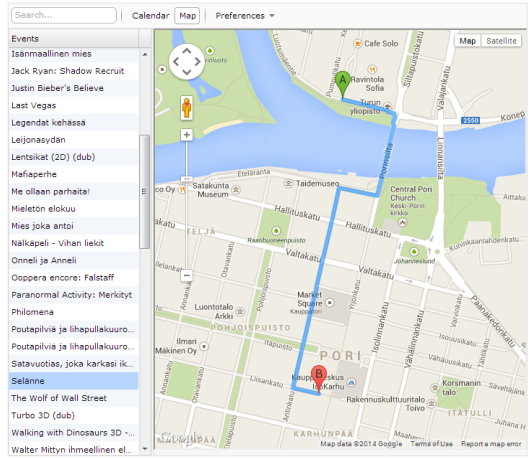

Figure 3. Map view

Fig. 3 shows the location of a selected event on a map using the Google Maps API [30] embedded in a w2ui panel. Showing a map and directions is often quite simple, but they can provide a lot of additional value for the end user. Most of the online map services provide JavaScript APIs, which can be used to show a location on a map and provide guidance to the location from the user's current location [31] on foot, by car or by public transportation. Unfortunately, in most cases the directions are not turn-by-turn capable and cannot as such be used to navigate directly to the target location except by continuously re-calling the mapping service, which can be very resource-intensive. To enable turn-by-turn navigation, the use of a native map application found on a mobile device is often required. If more complex map operations are required, the use of a native map application may be a better choice than a web-based client. In addition to extended functionality, native applications generally perform much better on mobile devices when compared to JavaScript web applications with similar functionality.

VI. DISCUSSION

This paper presented an event calendar service, which can be used to show data retrieved from various data sources, and provided some technological insight on how to implement the aforesaid service. Nevertheless, there remain several problems with the current implementation, and these problems are not strictly related only to the application in question. As mentioned above, the newer database technologies have not yet been fully utilized, and because of this it is often hard to find ready-made solutions for the problems. This is especially true when compared to traditional relational databases – several good books have been written and best practices developed to solve various implementation problems related to relational databases. The lack of best practices naturally creates problems for adapting new technologies, but on the other hand it makes the subject an excellent research topic. When adapting to a new technology, there is also a great temptation to abandon the old models and use new technology for every use case – even when it is not appropriate to do so. For example, it is possible to use a

key-value database or document database to map relationships, even though that is not what they are designed for, while a relational database would be a more appropriate choice.

Another problem is finding the event sources from the Internet. It is relatively easy to find listings (e.g. [32]) of usable event APIs by using a simple web search, but utilizing the sources requires implementing parsers for each of the various APIs. Implementing a parser is generally a straight forward task, but it does require one to study the API documentation and perform some programming work. Another option is to search events from popular web pages such as Facebook or Google Calendar – or use the APIs provided. The availability of events is dependent on the services used, and extracting the events may require special permissions from the publishers of the events whether they are the users of the service or the service operators. In practice it may be better to use the service APIs directly instead of crawling the web in hope of finding up-to-date event details in sensible format is difficult. Also, in many cases the online calendar applications are dynamically populated with, for example, JavaScript, and can be very challenging to parse. Extracting content from dynamically created web pages has been researched by e.g. [33] and [34].

In addition to purely technical problems, there are also other issues related to implementation and usage. In many cases it is often difficult to figure out whether the data in question is actually *open* data or merely *openly* available [35], [36]. This can be illustrated by using the example chosen in this paper. If the cinema does not show copyright notices or present license agreements on its web page nor are there any present in the provided data, does that mean that the data is truly open and free to use? Should permission be asked from the cinema owner, or maybe from the company who made the movies, or even from the company who made the local translations of the movie descriptions? In our use case one might assume that showing the movie details and showtimes on a public calendar is only good publicity for the cinema and a way to advertise in a way the cinema did not think of, but legally this may not be the case.

Also, many open data sources do offer data freely, but with certain limitations. The user may have to register first, or maybe the data cannot be shown openly and freely in all possible use cases. Finding these license agreements is not always a simple task. The service presented in this paper mostly ignores these issues by being a technology demo, but when designing an actual implementation, which will be a publicly available service, one should be careful to check that the apparently free and open data is truly free and open.

VII. Summary

This paper presented an event calendar service, which can be used to visualize open data. The overview of the service was presented, and the chosen technical solutions were explained. The issues encountered while implementing the service were also dealt with briefly. The pilot application proved that the chosen technologies were promising candidates for future studies. In the future, our research will continue to focus on the new database technologies. We will also continue to improve the calendar implementation, with a possibility of running a user trial on a local summer festival. It was also found that there is a need for a separate study on the issues concerning the licensing of open data.

References

[1] Tampere University of Technology, AVARAS Regional Initiative to Reveal the Opportunities of Open Data, https://www.tut.fi/avaras/?page_id=424. Retrieved February 24th, 2014.

[2] Open Data TRE, Open Data TRE, http://www.hermiagroup.fi/opendatatre. Retrieved February 24th, 2014.

[3] Helsinki Region Infoshare, Helsinki Region Infoshare | Open regional data, http://www.hri.fi/en/about. Retrieved February 24th, 2014.

[4] R. T. Fielding, Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000.

[5] Wikipedia.org, Representational state transfer, http://en.wikipedia.org/wiki/Representational_state_transfer. Retrieved February 24th, 2014.

[6] The Apache Software Foundation, Apache Tomcat, http://tomcat.apache.org. Retrieved February 24th, 2014.

[7] GoPivotal Inc., Spring Framework, http://projects.spring.io/spring-framework. Retrieved February 24th, 2014.

[8] E. Redmond and J. R. Wilson, "Seven databases in seven weeks," O'Reilly Media, ISBN 978-1-934356-92-0, 2012.

[9] The Apache Software Foundation, Apache Lucene - Apache Solr, http://lucene.apache.org/solr. Retrieved February 24th, 2014.

[10] Oracle Corporation, MySQL, http://www.mysql.com. Retrieved February 24th, 2014.

[11] The Apache Software Foundation, Suggester - a flexible 'autocomplete' component, http://wiki.apache.org/solr/Suggester. Retrieved February 24th, 2014.

[12] Neo Technology, Neo4j, http://www.neo4j.org. Retrieved February 24th, 2014.

[13] I. Robinson, J. Webber, and E. Eifrem, "Graph databases," O'Reilly Media, ISBN 978-1-44935-626-2, 2013.

[14] GoPivotal Inc., Spring Security," http://projects.spring.io/spring-security. Retrieved February 24th, 2014.

[15] MongoDB Inc., mongoDB – HTTP Interface, http://docs.mongodb.org/ecosystem/tools/http-interfaces. Retrieved February 24th, 2014.

[16] The Apache Software Foundation, SolrJ, https://wiki.apache.org/solr/Solrj. Retrieved February 24th, 2014.

[17] GoPivotal Inc., Spring Data, http://projects.spring.io/spring-data. Retrieved February 24th, 2014.

[18] E. Ort and B. Mehta, "Java Architecture for XML Binding (JAXB)," http://www.oracle.com/technetwork/articles/javase/index-140168.html, 2003. Retrieved February 24th, 2014.

[19] Google Inc., google-gson, http://code.google.com/p/google-gson. Retrieved February 24th, 2014.

[20] World Wide Web Consortium, "Web Storage," http://www.w3.org/TR/webstorage, 2013. Retrieved February 24th, 2014.

[21] Finnkino Oy, XML Services, http://www.finnkino.fi/xml. Retrieved February 24th, 2014.

[22] w2ui, JavaScript UI – w2ui, http://w2ui.com/web. Retrieved February 24th, 2014.

[23] The jQuery Foundation, jQuery, http://jquery.com. Retrieved February 24th, 2014.

[24] A. Shaw, FullCalendar - Full-sized Calendar jQuery Plugin , http://arshaw.com/fullcalendar. Retrieved February 24th, 2014.

[25] Telerik, jQuery-powered framework for desktop-style apps and mobile sites | Kendo UI Web, http://www.telerik.com/kendo-ui-web. Retrieved February 24th, 2014.

[26] PrimeTek, PrimeFaces, http://primefaces.org. Retrieved February 24th, 2014.

[27] K. Stetz, CLNDR, http://kylestetz.github.io/CLNDR. Retrieved February 24th, 2014.

[28] Bic.cat, BIC Calendar, http://bichotll.github.io/bic_calendar. Retrieved February 24th, 2014.

[29] The Apache Software Foundation, ExtendedDisMax, http://wiki.apache.org/solr/ExtendedDisMax. Retrieved February 24th, 2014.

[30] Google Inc., Google Maps, https://maps.google.com. Retrieved February 24th, 2014.

[31] World Wide Web Consortium, "Geolocation API Specification," http://www.w3.org/TR/geolocation-API, 2013. Retrieved February 24th, 2014.

[32] W. Santos, "93 Events APIs: Eventful, Upcoming, Google Calendar," http://blog.programmableweb.com/2012/05/01/93-events-apis-eventful-upcoming-google-calendar. Retrieved March 28th, 2014.

[33] C. Duda, G. Frey, D. Kossmann, and C. Zhou, "AJAXSearch: crawling, indexing and searching web 2.0 applications," in Proceedings of the VLDB Endowment, New Zealand, vol. 1, issue 2, pp. 1440-1443, August 2008.

[34] S. Choudhary et al., "Crawling rich internet applications: the state of the art," in Proceedings of the 2012 Conference of the Advanced Studies on Collaborative Research, Canada, pp. 146-160, November 2012.

[35] Open Knowledge Foundation, Open Definition, http://opendefinition.org/od. Retrieved February 24th, 2014.

[36] Open Knowledge Foundation, Open Licenses Service, http://licenses.opendefinition.org. Retrieved February 24th, 2014.

Publication VI

Rantanen P., "REST API Example Generation Using Javadoc", Computer Science and Information Systems, ComSIS Consortium, ISSN 1820-0214, 2017. *Accepted for publication.*

Publication VII

Ahmad, I., **Rantanen P.**, Sillberg, P., Laaksonen, J., Liu, S., Forss, T., Malik, A., Nieminen, M., Shetty, R., Ishikawa, S., Kallio, J., Saarinen, J. P., Gabbouj, M. and Soini, J., "VisualLabel: An Integrated Multimedia Content Management and Access Framework for Personal Users", in Proceedings of the 27th International Conference on Information Modelling and Knowledge Bases (EJC 2017), Krabi, Thailand, June 5-9, 2017. *Submitted.*

VisualLabel: An Integrated Multimedia Content Management and Access Framework

Iftikhar AHMAD[a], Petri RANTANEN[b 1], Pekka SILLBERG[b], Jorma LAAKSONEN[d], Shuhua LIU[c], Thomas FORSS[c], Aqdas MALIK[d], Marko NIEMINEN[d], Rakshith SHETTY[d], Satoru ISHIKAWA[d], Jarno KALLIO[e], Jukka P. SAARINEN[f], Moncef GABBOUJ[a] and Jari SOINI[b]

[a] *Department of Signal Processing, Tampere University of Technology, Tampere, Finland*

[b] *Pori Department, Tampere University of Technology, Pori, Finland*

[c] *Arcada University of Applied Sciences, Helsinki, Finland*

[d] *Department of Computer Science, Aalto University, Espoo, Finland*

[e] *Lynx Technology Finland Oy, Tampere, Finland*

[f] *Nokia Technologies, Tampere, Finland*

Abstract. With the rapid growth of image and video data as well as the fast spread of user-generated content in social media and the cloud, it has become increasingly difficult for users to have efficient access and effective management of their digital content. In this paper we present a novel integrated open source multimedia content management and access framework, called VisualLabel, that enables smart photo services based on automated visual content analysis, annotation, search and retrieval using the start-of-the-art analysis back ends for services such as Facebook, Flicker. This paper includes a detailed descriptions of the high-level architecture used in the VisualLabel framework and proof-of-concept implementations of a front-end service along with three analysis back ends and a web client, all of which demonstrate the basic functionality provided by the framework.

Keywords. CBIR, content management, framework, REST, web client

1. Introduction

With the rapid growth of image and video data along with the fast spread of user-generated content in social media on cloud services (Facebook [13], Twitter [61] etc.), it has become increasingly difficult for users to have efficient access and management of their digital content. Multimedia data growth is prodigious and increasing every year. As the number of users of multimedia handheld devices (e.g. smartphones, tablets, etc.) is increasing and low-end (feature) handheld devices are getting equipped with better camera and Internet connectivity, user generated content is also growing. Therefore, most of the digital data in network drives and social media sites has been generated in the last two years [51]. New multimedia handheld devices combined with web technologies provide users with new forms of multimedia sharing such as story-telling and social networking. Personal multimedia content is not only stored in personal

repositories but also on social media platforms (Facebook, Twitter, YouTube [74], Flickr [19], etc.) and network drives (e.g. Microsoft OneDrive [40]). Finding an image of interest from the terabytes of data distributed on shared network drives and social media websites is a challenging research problem for scientists, engineers and users alike.

The use of annotated metadata helps in organizing multimedia items. Flickr, Facebook and YouTube have shown that manually assigning text to images and videos can be fun and helpful for search purposes. People can annotate their multimedia content so that it can later be used for a text-based search to find a specific media item. Manually assigning text to a multimedia item is not only subjective but also noisy at least and at most can be totally irrelevant or misleading [12]. Consequently, proper text processing is required to use this metadata. Some of this metadata may be automatically generated during image capture, such as date, time, device information and location. Additional metadata can be generated by using different state-of-the-art object detection and other image analysis algorithms.

Current Content-based Image Retrieval (CBIR) systems can be divided into two categories: stand-alone systems that cannot access social media sites or network drives and mostly cannot handle huge datasets [1]; and the CBIR as Web service systems used for organizing images [25], [43], [46]. None of these systems combine social media sites with network drives or online content storage providers for image retrieval or content management purposes. Most of these systems use low-level and either global or local visual features such as SIFT [36] or SURF as bag-of-word (BoW) [69] [73]. Due to the semantic gap between low-level visual features and high-level content in an image, retrieval performance can be poor when dealing with large datasets [12].

Some efforts have been made (Google [21], Yahoo [71] and Baidu [6]) to use the surrounding text for image search. Zhang et al. [78] propose iFind for Web based image search. Cortina by Quack et al. [43] combines metadata and content for image search. Unsupervised clustering has been studied by Wang et al. [68], Gao et al. [20] and Cai et al. [8]. These systems can perform well on small or medium-size domain specific datasets but cannot work well in very large datasets. S. Kiranyaz et al. [30] proposes collective network of binary classifiers (CNBC) framework to achieve a high retrieval performance. The CNBC framework basically adopts a “Divide and Conquer” type approach by allocating several networks of binary classifiers (NBCs) to discriminate each class and performs evolutionary search to find the optimal binary classifier in each NBC. To the best of our knowledge, none of the systems uses the surrounding text from social media sites along with image metadata to create ground truth and train the system for efficient CBIR.

Traditionally, CBIR uses the Euclidean distance or the cosine similarity distance functions on low-level features for image dissimilarity [12]. However, these functions may not be optimal for content-based image retrieval due to the semantic gap mentioned earlier. Therefore, there has been a lot of research into automatic feature extraction and the optimal distance calculation methods of the extracted features [12]. Krizhevsky et al. [33] and Wan et al [65] applied deep learning for content-based image retrieval. S. Kiranyaz et al. [31] proposes feature synthesis to improve the discrimination power of the features for efficient retrieval.

It is hard to teach what we see around us and how we perceive it to machines. Although visual similarity is more critical in the case of machine-to-machine interaction but for humans, semantic similarity is also important. There have been many efforts regarding machine learning to understand visual content. Recent

breakthroughs in artificial intelligence [33] [65], such as deep learning, provide new methods for content-based image retrieval. With the help of large-scale training data and huge computation power, it is possible to train deep models based on Deep Convolutional Neural Network (DCNN) [33]. There are a number of libraries available that provide pre-trained models for feature extraction.

CBIR is extensively used in remote sensing, satellite images, digital libraries, education, entertainment, medical images (X-rays, MRI etc.), face/finger print databases, map and personal content management. Nevertheless, there remains the question of how to enable this technology for the ordinary user [12] to manage multimedia items from social media along with personal repositories. In our study we mainly focus on personal content management although the technology can be used in all of the areas mentioned above. We have developed a novel and comprehensive open source content analysis framework – called *VisualLabel* – that provides multiple views of the same multimedia items shared on social media sites and network drives for user's content management needs. It automatically organizes the user's multimedia data by using state-of-the-art machine learning algorithms to detect faces and different objects, for example, in the user's personal collection of images. The framework can also utilize multiple external web and cloud services (such as Flickr), and offers the end-user a Representational State Transfer (REST) Application Programming Interface (API) for multimedia organization and retrieval.

The rest of the paper is organized as follows. Section 2 gives the background of the studies related to our work. Section 3 details the architecture of the VisualLabel framework. Front-end services as the main building block are presented in Section 3.2. In Section 3.3, we provide details about back-end systems and their contributions. User experience and the web client are presented in Section 3.4. Experimental results are presented in Section 4 and finally, we conclude the paper along with future research directions in Section 5.

2. Related Studies

Digital cameras have revolutionized photography and provided a large volume of inexpensive digital images and videos. Early efforts were mainly focused on metadata to organize images. Although early digital cameras did not produce that much metadata, manually assigning metadata is not only subjective but is also cumbersome [12]. Later, the research focus changed and content-based image retrieval was used for image retrieval, such as in VisualSeek [57] and WebSeek [58] developed by Smith and Chang and MUVIS [39]. These systems focus on color/texture region extraction in both the uncompressed and compressed domain. The color set concept is used in their several variations of color, texture and shape features. Instead of decompressing the existing compressed images to obtain the texture features, Smith and Chang perform texture feature extraction in the compressed domain, such as Discrete Cosine Transformation (DCT) and Discrete Wavelet Transformation (DWT). Wang [67] developed the SIMPLIcity system, in which categorization – such as graph versus photograph, textured versus non-textured – is applied to images, and different sets of features are used for each category. MUVIS was developed as stand-alone applications for content-based multimedia indexing and retrieval in multimedia databases. It supports a number of feature extraction modules for color, texture, segmentation and audio for multimedia indexing and retrieval, whereas Mobile-MUVIS [1] is an extension of MUVIS to

perform content-based multimedia retrieval on mobile devices. Local feature representations such as bag-of-words models using local feature descriptors (SIFT, SURF) are used for content-based image retrieval. Rui et al. [50] implemented the MARS system, which includes relevance feedback. MARS uses user relevance feedback to refine the query. VisionGo [37] integrates user feedback to meet the user expectation in the graphical user interface. Cuzero [76] targets a fast response to user queries through its web interface. Anaktisi [9] is a web solution based on a color and texture histogram for retrieving images. BRISC [34] is a content-based image retrieval system based on low-level features. Imprezzeo [25] provides a visual search service as a REST API. The web-based Mobile-MUVIS, Anaktisi and BRISC provide query-by-example or browsing the database. Retrievr [46] is based on Fast Multiresolution Image Querying. VIRal [28] is a content-based image search engine that provides query support for uploaded images, provided URLs or from the VIRal database. Windsurf [4] is content-based image queries with the emphasis on a region-based paradigm. PIBE [5] from Bartolini is an adaptive image browsing system. In addition, there are many commercial services such as Google [21], Yahoo [71] and Baidu [6].

All the above systems and frameworks mainly focus on feature extraction or learning and content retrieval only, whereas VisualLabel is a comprehensive framework for multimedia data management. It provides alternative ways to manage the media items by using different back-end systems with different capabilities. It specifies an interface for securely data exchange and considering privacy issues. Social media sites contain personal data of the user that individuals might want to share with friends and family members but might not share it publically. VisualLabel framework hide the personal information of the user and only pass the media items to the back-end. Back-end delete the media items after analysis and pass the analysis results to the VisualLabel server.

VisualLabel provide adaptive user interface for a wide range of devices. It displays automatically generated metadata during media capture and back-end generated to the users. User can provide feedback on the generated metadata and this feedback is provided back to the back-end for incremental learning. It also provides seem less integration to different social media sites, to collect multimedia items along with metadata (comments, likes etc.) and create context of a user activity.

3. VisualLabel Framework

A general overview of the VisualLabel framework can be seen in Figure 1. The main users can be categorized into four groups: clients, front end(s), back ends and content providers. The clients perform queries and browse the content enriched with metadata using the service provided by the front end and also use the services offered by the providers for adding content to the external services. One example of a possible client is the prototype web interface presented in section 3.4.

On a higher level, there is only a single front end – or front-end service – though in practice there might be several front-end servers depending, for example, on scalability and performance requirements. The front end works as the central point that binds the service framework together by synchronizing the content from the providers, creating tasks for back ends and providing client APIs for users. The front-end functionality is explained in more depth in section 3.2.

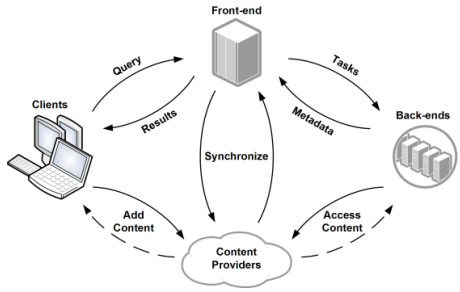

Figure 1. VisualLabel framework.

There is usually more than one analysis back end, either for load-balancing reasons or because of back-end capability limitations. In our case, the back ends have slightly or widely different capabilities – one back end could generate keyword tags based on the given image content, another could analyze the user's social media profile. The back ends utilized in our framework and their capabilities are discussed in more depth in section 3.3.

The last group consists of content providers. These are the external services used to host the user's content, such as image or video files, or the user's social media profiles – examples being Picasa [42] and Facebook. The users generally use the external service through applications or web interfaces provided by the services themselves and the VisualLabel framework accesses the content hosted on the services by utilizing REST APIs or similar interfaces provided by the services.

Another novelty of the VisualLabel framework are the efficient interfaces of the proposed architecture. The API model for VisualLabel is shown in Figure 2. In the figure, Content Provider Service APIs are implemented by the external content provider services. Client APIs, Content Provider APIs and Server APIs are implemented on the front end. Mediator APIs – if required – are implemented on the back end or on some intermediate service. Finally, Back-end APIs are implemented by the analysis back ends. This architecture hides the algorithms and communications protocols from the end user and thus provides a more efficient and simple to use interfaces.

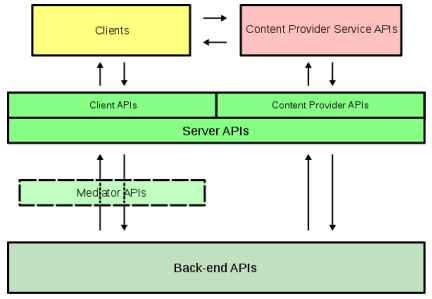

Figure 2. API model.

Clients are the end-users of the service and use both the APIs offered by the providers (Content Provider Service APIs, in Figure 2) and the Client APIs. Content

Provider Service APIs are defined by the content providers and their counterparts on the front end are the Content Provider APIs, which are in our case callback methods for various authentication methods, such as Google's or Facebook's [14] OAuth2 [7]. Using public REST APIs is generally the easiest solution and most companies offer pre-made libraries for commonly used programming languages (such as Java and JavaScript) that help in implementing access to their services. The downside is that these APIs cannot be modified to match the needs of the user – in this case, the front-end service – and the implementation is dependent on the content provider's specification. Whether the limitations enforced by the content providers matter or not depends on the use case, but it does imply that the front-end implementation must be modified when breaking interface changes occur at the provider's end.

Client APIs are a RESTful service, which contains the basic interfaces required for user management (e.g. account creation, user login, external account connectivity) as well as the more specific interfaces that enable the modification and removal of the generated metadata, content queries, and feedback or ranking of the generated metadata. The choice of using REST [17] [18] for the client-side interface is based on three reasons: firstly, REST is becoming a de facto method for building client interfaces, mainly because of its adoption by major software companies; secondly, because it offers a simpler and easier-to-learn means to access resources; and thirdly, there is a slight performance gain when compared to the traditional web service oriented way [2] [16] [23].

Server APIs contain both the callback methods for Back-end APIs and the utility methods for checking status information for ongoing tasks. The status information can contain the same information as that delivered to the back ends, such as media details and information about the runtime progress of the task – i.e., which back ends are participating in the task and whether or not the analysis has been successful. Error cases are strictly internal to the service, and from the user's point of view, a failed analysis simply results in a smaller amount of metadata.

Back-end APIs contain the methods necessary for scheduling new tasks for analysis and can be considered to be the counterparts for Server APIs. Mediator APIs may be needed in case an external back end is used when the back end's implementation cannot be modified to match the specification dictated by the proposed VisualLabel framework. The Mediator APIs can be on an entirely separate middleware server or hidden behind the back-end interface implementation.

The communication between the Server and the Back-end APIs – whether Mediator APIs are present or not – should be routed through more secure channels either by using stronger authorization rules or establishing a demilitarized zone (DMZ) between the components. Unlike client and content provider connections, which are individually authenticated, the back ends have a much higher level of access to the front end and vice versa. User credentials are never delivered to the back ends, but the tasks often contain static Uniform Resource Locators (URL), which can both ensure user anonymity on the back-end analysis, and it can be used to directly access the user's content, bypassing the user authorization process. Additionally, even though user credentials are not transferred between the front end and the back end, the tasks themselves may contain sensitive content, such as the status messages posted by a user on a private social media account.

The Server and Back-end APIs do not fully follow the RESTful principles, and the service description is closer to what is called a REST hybrid [47]. In particular, the resource representation does not strictly adhere to the resource-oriented model even

though the tasks –i.e., the workloads delivered to the analysis back ends – can be considered to be resources for the service. The tasks are identified by Uniform Resource Identifiers (URI), though the links are not cacheable because the task content is not guaranteed to be static. Removal of a media item would cause the item to disappear from the task or the addition of more metadata could cause the task to be updated. In either case, the identifier – and the links remain the same. Also, the communication includes certain state-fullness in the form of a front end keeping track of the remote tasks and their progress. Thus, by nature, the APIs are more closely related to the traditional Remote Procedure Call (RPC) interaction.

3.1. Use Cases

The primary goal of the proposed VisualLabel framework is the novel integration and utilization of the numerous CBIR solutions in combination with the existing services – such as social media web sites, network drives, and other content storage services – which are commonly use today. Existing literature contains no such comprehensive framework. Our study also shows that users feel that there is a need for better tagging services and richer features. VisualLabel aims to provide the high-level architecture and required APIs for integrating multiple analysis engines (back ends) for a feasible end-user product. Additionally, we provide a proof-of-concept implementation based on an open source implementation [64].

To enable the desired functionality and the interoperability of the back ends within the VisualLabel framework, certain common use cases need to be defined. The use cases are listed in Table 1. The basic assumption is that the user has an existing account on one of the supported external services (e.g. Facebook [13], Picasa [42]), or that the user can create an account (use case 1, Table 1). The implementation for the first use case is not important in terms of VisualLabel and is fully up to the choices of the third party content providers.

Table 1. VisualLabel use cases.

Use case #	Use case name
1	Creation of an external account (e.g. social media or network drive account)
2	Account creation on the VisualLabel framework (service)
3	Content upload to the external account
4	External account synchronization and task construction
5	Back-end task analysis and metadata generation
6	Content browsing
7	Content-based query operations

Use case 2 (Table 1) can be realized either by using the framework's registration forms to create a new user and then manually connecting the desired external accounts to the service or by simply creating a new account using the credentials of the external account. Both approaches are common on many popular Web applications. Similarly, to use case 1, case 3 can be fully implemented by third party content providers. Use

case 3 could also be integrated directly into a potential Web or a mobile application. Cases 4 and 5 can be considered two parts of the same operation. The purpose of these cases is to process the connected external accounts for usable content (case 4), which could be delivered to the analysis engines for further analysis (case 5). The metadata generated by the analysis is combined with the content found from the user's other accounts. This enriched content can be shown to the user to help the user organize and browse their media collections more efficiently (cases 6 and 7).

In principle, the order execution for the use cases follows the numerical order presented in Table 1, though in a simplified fashion, with the initial use case being at the top. In practice, many of the cases can be repetitive or cyclic in nature. For example, the user can perform any number of query operations (case 7), or return to connect to another external account (case 2). Additionally, account synchronization (case 4) and back-end analysis (case 5) should be performed periodically to ensure that the modifications or additions made by a user to the connected accounts are properly updated.

3.2. VisualLabel Front End

The purpose of the front-end service – as shown in Figure 1 – is to integrate a native client or a Web client with the back-end analysis engines, and to provide feasible interfaces for the realization of the use cases illustrated in Table 1. There are three important tasks performed by the front end: user management; task generation; and query processing.

For user management, in addition to more traditional functionalities (e.g. user account management, service login etc.), also includes more complex operations such as external account synchronization and content redirection to clients and back ends. The user can connect any number of external accounts to the service either by first creating new credentials for the service and connecting the external account to these credentials, or by registering directly with external credentials, for example, by using a Google or Facebook account. In either case, the connected account will be periodically synchronized with the front-end service. The synchronization operation consists of retrieving and indexing metadata from the user's external accounts. For social media accounts the metadata can be anything from basic user details to status messages that the user has posted. For photo, video and audio content the metadata always contains the information needed to access the content, in addition to other content-specific metadata. In fact, the actual files hosted by the content providers are not transferred to the front end or stored anywhere on the VisualLabel service other than temporarily on the back ends for the duration of the content analysis. Not storing the files helps to minimize the potentially massive storage space requirements and also lowers the bandwidth requirements since the files are not directly hosted by the service.

The indexed metadata is used to construct analysis, feedback and search tasks for the back ends. The task-based approach for content analysis including the feedback functionality [44], in this section we provide a short description of the task types used and the task functionality in general.

Analysis tasks are created automatically upon account synchronization or whenever the user submits new content for analysis. The responses to analysis tasks contain automatically extracted metadata, such as faces or keyword tags detected in the content. The extracted metadata is appended into the front end's index and can be modified by the user, updated by the back ends or used in search queries.

Feedback tasks are always created after the user has made changes to the media content. The feedback task creation can be indirect or direct. Indirect task creation can happen after the user modifies the indexed metadata or after account synchronization when certain changes, such as content deletion or addition, are detected. In general, if a user repeatedly makes the same changes into automatically generated content, this can be assumed to mean that the metadata is incorrect or simply something the user does not prefer. Direct feedback is created when the client software takes advantage of the provided feedback interface. The interface can be used to rank the extracted metadata, and to give feedback on the quality of the search results.

Providing the back ends with information about the actions of the user can help the back ends improve their analysis results in the future. How the back ends take advantage of the provided feedback is solely defined by the back ends themselves, and is not specified by the framework. Similarly, to the case of metadata generation for analysis tasks, the feedback processing implementations are specific to each back end and to the analysis methods chosen by the back ends. The front end and the framework itself are not interested – at least in principle –in how the back ends produce their results, but in what kind of results the back ends can provide.

Search tasks are created when the user performs a query, which cannot be directly resolved by the front end. For example, if the user uploads an external reference photo not known by the service to be used for the search of similar photos, the query cannot be executed simply by looking at the indexed metadata on the front end, but the photo must be delivered to the back ends for further analysis. From the performance point-of-view, the search tasks can be considered to be more demanding for the back ends than analysis or feedback tasks. The reason is that there is a certain timeframe within which the back ends must respond in order for the functionality to remain usable. For search tasks, users often feel the service is not responsive or too slow if the operation takes more than a few seconds. The analysis and feedback tasks can be processed at the pace and order decided by the back ends as long as the results are returned within a reasonable time period.

All tasks are strictly using VisualLabel framework syntax, and although the metadata used in the task can be in different format but if necessary, the metadata retrieved from other sources is translated to the common format. The simple reason for this is to allow the back ends to operate on a single format without knowledge of the various external services and formats specified by these services. There is one practical problem related to this approach. In some cases, it might be difficult or cumbersome to translate all content from the external services to the VisualLabel format, or to implement a conversion module. In practice, some information may get lost in the conversion. It is worth noting however, that not all information about the external source is removed. The tasks – and the task items – contain identifiers, which denote the origin of the data. The reason for this is that even though two services may belong to the same service category, analysis of the two data sources may differ. For example, although both Facebook and Twitter are social media websites, and with status messages, likes, friends, and followers they do contain similar elements, it may be advantageous for the analysis engine to know from which social media service the data originates.

The task and metadata format for the framework are Extensible Markup Language (XML), but could as well be, for example, JavaScript Object Notation (JSON). There has been an ongoing trend toward the adoption of JSON instead of XML, the main reason often quoted being a small performance increase [2] [48], though in practice the

service implementation – and the format parser implementation – dictates most of the available performance gains when using reasonable sized payloads, with the choice of format being secondary.

3.3. VisualLabel Back Ends

A back end is any system or service that implements the defined back-end API (see Figure 2). Back ends are the systems that analyze the task content provided by the front end and either generate new metadata or participate in extended query operations (such as query-by-example). This section presents the three content analysis back ends with different capabilities utilized in the implementation of the VisualLabel framework.

3.3.1. Cloud MUVIS

Content-based multimedia retrieval has been an active area of research for decades, with applications in a range of areas such as personal visual content management, security, etc. Cloud-MUVIS (CMUVIS) is used for content-based image retrieval in the VisualLabel framework. It is a cross-platform, robust, and scalable computing platform for content-based multimedia retrieval. CMUVIS is a big data processing system with a small footprint, using a distributed computing model where small portions of a large database are processed in parallel. However, as the data volumes are increasing at an enormous rate, a huge amount of processing power for analysis is required. CMUVIS uses burst mode computing to process data that utilizes underlying resources (Cores, Memories and Disk access) efficiently. CMUVIS provides a Fex Application Programming Interface (APIs) for feature extraction and content-based query operations. As shown in Figure 3, the CMUVIS framework processes different data partitions in parallel.

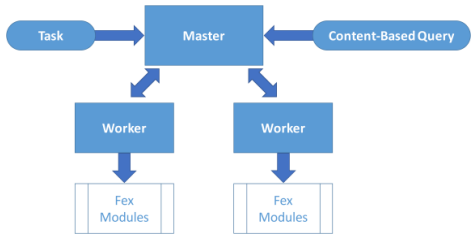

Figure 3. Cloud MUVIS system.

The main goal of the VisualLabel framework is to organize personal visual content (images and videos) that contains human faces. Therefore, it is important for a back-end system to detect faces and recognize them. For facial features, we use a recently proposed feature selection algorithm, namely a cardinal sparse partial least square algorithm [77]; it selects a sufficiently large number of features, with the most informative and discriminant features given that can achieve good accuracy when used with linear classifiers.

CMUVIS as a back-end system provides a content-based search of the multimedia items that consist of the following model. As shown in Figure 3, tasks on the left side are processed offline and features are extracted, and online query is performed on the right side.

In offline processing,

1. The image is checked for a face(s), and it/they are registered
2. Objects in an image are detected and labeled using deep learning
3. Convolutional neural-network features are extracted from the image for similarity distance.

In the online query operation first a face is detected in the incoming query image; if there is a face, then all the images with a similar face are found. Otherwise, deep convolutional features are extracted from the query image and compared against all the images in the database, thus a sorted list is created based on a similarity score.

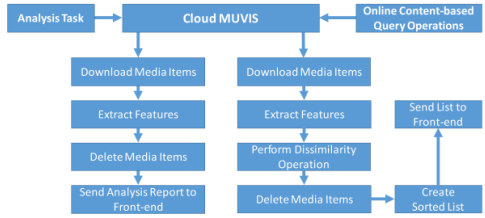

Figure 4. Cloud MUVIS offline and online operations.

Figure 4 shows the offline analysis task (on the left) and online content-based query operation sequence (on the right).

3.3.2. PicSOM

The PicSOM back end is used for automatic content-based annotation of image and video content. These annotations can be of two types: keyword-based and sentence-based. The keyword annotation has been developed mainly for the video domain, especially for use in TRECVID Semantic Indexing (SIN) tasks [26] [41]. Special attention has been paid not only to the accuracy, but also to the scalability and speed in the content classification stage. The accuracy [32] has been obtained by using late fusion of multiple (up to 10-40) outputs from SVM detectors created with different spatial pyramids of deep convolutional neural network (DCNN) [29] [56] [59] [75], BoW SIFT [53] and MPEG-7 Scalable Color features. The speed, on the other hand, follows from the utilization of homogeneous kernel map linear SVMs [15]. As all the features we used are extracted in frames (either in all I-frames or in selected key-frames), the final fusion step combines the frame-level detection results with the shot level by using max pooling. From the TRECVID SIN training data, the PicSOM back end has a basic set of 350 visual keywords it can recognize from both images and videos.

A sentence-based annotation of images and videos has recently been included in the PicSOM system when large image [35] and video [49] [60] databases have been made available with sufficient numbers of visual and textual samples for sentence generator training. In both modalities, sentence generation is based on using DCNN features as inputs to the Long Short-Term Memory (LSTM) [24] [29] [63] recurrent neural network. In the image domain, we use multiple features to train different sentence generators, which are then used to evaluate each other's outputs in order to select the most accurate caption for the input image. In the video domain, in addition to

the DCNN features, we also use dense trajectory features [66]. The former is used as inputs to 80 content classifiers trained with the COCO categories. The latter, together with the outputs of the COCO category classifications, are fed as inputs to the LSTM network [24]. The information flow and processing stages are illustrated in Figure 5.

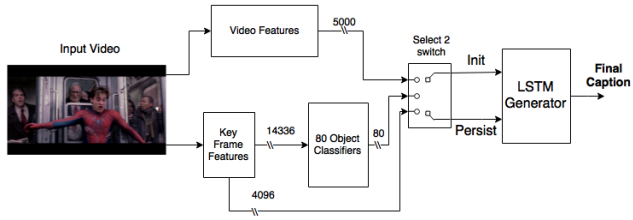

Figure 5. Block diagram of the PicSOM back end in video caption generation mode [54].

In the VisualLabel framework, the PicSOM back end is able to collect sets of images that share a common keyword (i.e., a class label or tag) and then extract DCNN and other features from them. A linear SVM classifier can then be trained and used to generate corresponding keywords for any images and videos the front end sends to the back-end. For sentence-based annotation, the back end can take in images and videos, extract the required DCNN and dense trajectory features, and generate a caption to describe the input visual content.

3.3.3. Social Media Tag Extractor

Visual labels or tags are generally of three types:

1. Content Labels: labels generated from direct analysis of visual content to identify types of physical objects or activity/actions in images or videos through learning from datasets labeled with object concept terms.
2. Context Labels: Location and space; Time; Social, Cultural and Personal Life events (Social media (face book); Domain and Topics.
3. Subjective Quality Labels: Opinions or sentiments.

Content-based visual labels are mostly "functional", as they indicate what the image is and what it is about, using terms from general or domain specific formal concept taxonomy, following standard schemes for classification/tagging. They are professionally defined, accurate, consistent, controlled vocabulary, restrictive and static in nature; therefore, they can easily suffer from coverage and scaling problems.

For images with little textual information, annotation using visual content is a natural solution. However, formal concept and object labels from visual content analysis are much more machine-friendly than user-friendly. There is a need to explore more relevant and user-friendly visual labels that include "context" descriptors. It is our intention to leverage the potential of social media resources for the benefit of visual annotation, to make use of social context information in facilitating image organization, access and utilization.

Context-based labels can be personal, social, or domain- and topic-dependent. Social tags and text associated with images and videos on popular social media sites are sources of rich semantic clues and context clues for broader indexing. They are author-given and may be general or personal, subjective or objective. They are user-

generated metadata with the user's choice of terminology, flexible, unstructured, and with no vocabulary control, informal with non-hierarchical flat organization. Therefore, there can be lots of noise, errors, irrelevance, redundancy, and ambiguity, with a varying level of granularity. They are emergent in nature and constantly evolving [62].

Given an input image, our goal of automatic image annotation is to assign a few relevant text keywords to the image that reflect not only its visual content but also its social context. To explore the possibilities of leveraging social media content as a resource for visual labeling, we have developed a tag extraction system that applies heuristic rules and a TF-IDF term weighting method to extract image tags from associated tweets. The system retrieves tweet-image pairs from public Twitter accounts, analyzes the tweet to extract a number of items as candidate tags: named entities (location, people and organization), hashtags, keywords and phrases (TF-IDF weighted words, frequency weighted bigram and trigrams). The ranking algorithm examines the extracted items and removes any redundancy between named entities, hashtags and ngrams (representing topics). Then post-processing is done to remove noisy tags, currently based on heuristic rules, which will later on be extended with more advanced methods. The system output will be a balance of different types of tags, depending on the targeted usage. The system also contains components to detect the language of the tweet and automatically translate a non-English tweet into English to be analyzed. The extracted tags are then translated back into the original language. The pre-processing of Twitter text is straightforward; the only requirements are, to pay special attention to some special characters and add to the stop word lists. Emoticons are removed for the time being, as we only target content and context labels, not sentiment labels.

For named entity recognition, we have applied the Stanford Named Entity Recognizer, which identifies the names of people, places, and organizations quite satisfactorily. Other types of proper nouns, e.g. names of products, books, magazines, movies, sports, and other events and activities can often be identified in the hashtags or key phrase list. Keyword and key phrase extraction help select a small set of words or phrases that are content bearing. As tweets have a very short text body, we have taken the simplest method for word weighting: TF-IDF for individual word weighting.

N-gram extraction allows to extract multi-word or phrase labels to describe any entities, topics, and activities better. Each of the ngrams is weighted by adding together the unigram weights for each word the ngram contains. Ngrams that start with or end with stop-words and punctuation are omitted. For language translation, we make use of the freely available Yandex [72] translation resource to translate content of any language into English.

The extracted tags are a mixture of concepts at different levels, and can sometimes overlap with high level concept and formal tags, but in most cases this does not happen. Hashtags can overlap substantially with Named Entities and text tags (key words and ngrams). We use heuristic rules for the first layer post-analysis and processing of the tags: (1) keep all hashtags; (2) named entities in hashtags are considered more relevant, so they get higher priority; (3) Ngram weighting is adjusted by individual word TF-IDF weighting; (4) we consider variety a necessity and priority of a good tag set, to include location, time, organization, people and topics.

Removing noisy tags is important. As expected, the extracted tags are of varying levels of granularity. There is a lack of consistency and lack of relationships between the tags when comparing with content-based labels.

A Facebook profile summarizer that contributes to the larger VisualLabel framework through summarization of Facebook content to extract information on the

user's hobbies and interest is also developed (Figure 6). By utilizing statistical text summarization techniques and heuristic methods, Facebook profiles and timeline content are analyzed to determine the most representative terms indicating people's hobbies and interests. The extracted key terms/phrases could be a different set of useful context indicators. They may also find use in other applications. A baseline system was developed as a simple, generic system that applies heuristic rules and the TF-IDF term weighting method in determining the most representative terms indicating hobbies and interests.

Figure 6. Facebook profile content analysis.

A pilot test was done to collect feedback from users concerning the perceived usefulness of the extracted tags. The baseline system was then extended to include new functionality to help set limits on the scope of relevant content, extract Named Entities, use of predefined dictionaries to identify even low-scoring hobbies and interests, and use of machine translation to handle content in multiple languages.

Our pilot test with a limited amount of English-dominant user profiles shows a 43% average of useful tags, with the highest being 55% and lowest at 25%. Keyword extraction seems more useful in the analysis of social media content than a sentence-based summary, due to the fact that social media content is not structured as professional texts. Named Entity Recognition becomes effective when combining state-of-the-art tools with semi-structured Facebook profile content.

3.4. Web Client

There is no built-in or default client to access the VisualLabel services as a whole, but we created several smaller HTML5-based prototype applications for demonstrating the features of the system. One of the prototype applications is called "Smart Photo Service" [3] [52]. The application (in Figure 7, Figure 8 and Figure 9) is used to display the photos of the user retrieved from various content providers (e.g. Picasa, Facebook and Twitter). It enables the user to browse, search, and tag photos using a web browser. The main goals of the application are: to explore the feasibility of keywords generated by the underlying back-end services; to study the user experience associated with photo tagging; and to utilize HTML5 technology to determine compatibility and usability on a broad range of devices and form factors (e.g. tablets, mobile phones, desktops).

One of the most important features of a modern web service is support for cross-platform compatibility. In general, the simplest way to achieve similar functionality on multiple devices is to implement an HTML5 and JavaScript based web application – similar to the one presented in this section. There can be minor differences in the behavior of the web user interface caused by hardware variations of the devices (screen

resolution, input devices, etc.) and by differences in the capabilities of the mobile web browsers. Both of these issues can be mitigated by using one of the commonly used cross-platform JavaScript libraries, such as JQuery [27]. As an illustration of cross-platform use the main view of the prototype client can be seen running on multiple devices in Figure 7 (left) on the Android operating system on Google Nexus; the Maemo software platform on the older Nokia N900; the Windows Phone operating system on the Microsoft Lumia 532; and Sailfish OS on the Jolla smartphone. For comparison, The Figure 7 (right) also shows the same user interface on the Google Chrome desktop web browser. As can be seen in the figure, the user interface is virtually identical on each device. In our case the web prototype implementation directly uses the client-side REST API methods using AJAX. Implementing the web client in this way makes it easier to modify the client application as there is no strong coupling between the service and the user interface. Additionally, the same REST API could be used to implement a native client on any device if so desired, for example, if a more native look and feel is needed, or if the web browser on the device is not sufficient in terms of features or performance for processing the web page.

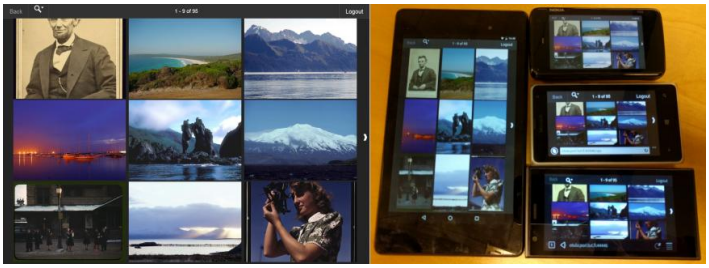

Figure 7. Web client running on a desktop web browser (left) and on several mobile devices (right).

The photos in the main view (Figure 7) are arranged in a 3x3 grid, and by swiping left or right the user can page through the photos. The arrow buttons located on the left and right can also be used for navigation in lieu of swiping. From this view, the user may click any photo to get a full screen view of the clicked photo, open a search overlay (as seen in Figure 8) by clicking the magnifier glass icon on the tool bar, or initiate a content-based search (screen shot on Figure 9) by long-clicking a photo.

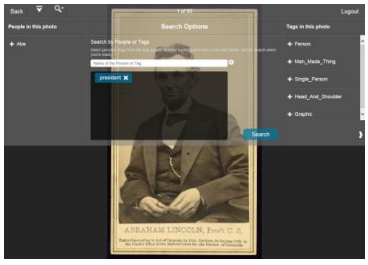

Figure 8. Web client search view.

The search overlay of the prototype application (Figure 8) displays the people tags on the left side, selected search terms in the middle, and keyword tags on the right. The

tags may contain keywords generated and suggested by an analysis back end, tags obtained from the originating content provider, or tags added and edited by the user.

The free text search input field in the middle is aided by an autocomplete feature, which suggests similar strings of text which have appeared before in any tag of the user's photos or in content that has been extracted by social media content analysis. The goal is to provide the user an overall impression of what can be found among the photos, and the user can utilize any combination of the provided search options (free text, tags and people search).

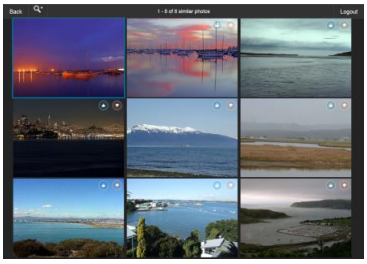

Figure 9. Web client content-based search results.

In the similarity search mode, the chosen photo is processed by back ends in order to return visually similar photos. The photos used in the similarity search are generally from the authenticated user, but it is also possible to use public photos from other users. The results of the similarity search can be seen in Figure 9. The blue-framed photo in the top left-hand corner is the reference photo, i.e. the photo that the user wanted to use in the similarity search. The result photos are laid out in a 3x3 grid in the same way as in the main view. There are only eight results in the example, but should there be more, the same navigation features, as available in the main view, can be used to browse through them. The user may also invoke a new similarity search from this mode by long clicking any photo.

4. Experiments

To validate the feasibility of the proposed VisualLabel framework, two types of experiments were performed. Because of the inherent complexity of the overall architecture, it is difficult to test out all components with a single test or benchmark. We decided to run a user study to ascertain the users' tagging habits, and to verify that the user interface design and logic is practicable from the users' point of view. Additionally, the back-end designs and implementations were tested separately to validate the correctness of the generated tags. The performed experiments are explained in the following subsections.

4.1. User Study

It is important to understand how people manage their digital photos on different devices. In particular, handheld devices are not only limited in their input and output methods and display properties but different devices also have different hardware and

software features. To understand the actual needs and the current behavior of users regarding the tagging of personal photos, we carried out a user study [3]. The user study was based on the task analysis approach together with pre-test and post-test questionnaires, and was carried out with 15 participants at the Usability Lab of Aalto University. The participants came from various backgrounds including students, entrepreneurs, and software engineers. Out of the 15 participants, ten were male and five were female. Nine of the participants were aged between 18–24 years, five participants were between 25-34 years old, and one was 39 years old. In each session we discussed with the users their current photographic practices in general, and more specifically photo tagging. The actual task analysis was carried out on a semi-automated photo tagging application running on the VisualLabel framework. Each participant carried out nine different tasks on a tablet device that addressed navigation, searching, and tagging, designed to study actual user interaction and behavior.

The study results give a clear indication that tagging personal photos is not actively used by most of the participants as only a few of them (2 participants) frequently engaged in tagging activities. The most important reason mentioned by the participants was the effort required to carry out tagging on a huge number of photos. Some of the participants did not feel the need for tagging their photos; on the other hand, some of them were not aware of the benefits of tagging, as they had never become familiar with the practice. After carrying out the tasks with the web client, the respondents considered the tagging activity highly beneficial for organizing and searching their personal photo collections. Based on the interactions and the input from the users, it became evident that usage and adoption of tagging features can be enhanced by increasing the visibility of tagging and by bundling social elements such as linking and sharing photos on social networking sites into web applications. Moreover, associated features such as highly relevant and comprehensible automatic tag suggestions may also improve the adoption of tagging features.

4.2. CMUVIS Back End

CMUVIS back end uses convolutional neural network features [29][54] for content-based multimedia retrieval. As there are not enough images to train a big network from a single user or a few users, we therefore use a pre-trained network for feature extraction. In Figure 10, the first column shows the query image and subsequent columns show the retrieval results.

Figure 10. CMUVIS query results.

CMUVIS evaluated the relevance of VisualLabel by participating in the Microsoft Bing International Challenge on Web Image Retrieval and our two submissions were ranked second and fourth positions. Contesting systems are requested to generate a

floating-point score on each image-query pair that reflects how relevant the text-query could be to describe the given image, higher score indicating higher relevance. The core of the CMUVIS system [45] is formed by a text-processing module combined with a module performing PCA-assisted perceptron regression with random sub-space selection (P2R2S2). P2R2S2 uses features [54] as a starting point and transforms them into more descriptive features via unsupervised training. The relevance score for each query-image pair is obtained by comparing the transformed features of the query image and the relevant training images. We also use face and duplicate image detection to boost our evaluation accuracy. Our system [45] achieves 0.5099 in terms of DCG25 [45] on the development set and 0.5116 on the test set [45].

4.3. PicSOM Back End

Two series of experiments were carried out with the PicSOM back-end system in order to evaluate its applicability to the use cases of the VisualLabel framework. First, the precision of the produced keyword annotations for video shots was evaluated in the TRECVID 2014 Semantic Indexing Task. Second, the accuracy of the video shot captions was assessed at the Large Scale Movie Description Challenge 2015, as described in the subsections below.

4.3.1. Video Shot Keyword Annotation

The mean extended inferred average precision (MXIAP) measure is used in the TRECVID Semantic Indexing Task (SIN) to assess the query results submitted by the participants of this annual evaluation. Figure 11 displays the MXIAP results of the 75 submitted results in the TRECVID 2014 SIN task with black and gray bars, the former being the PicSOM team's submissions and the latter belong to other groups. The MXIAP values are average results of 30 queries for different keywords such as "running", "stadium", "beach" and "hand". For each keyword, the participants submitted a maximum of four runs with a maximum of 2000 video shots from the evaluation data, in decreasing order of relevance to the keyword or visual concept in question. For the individual submissions, the PicSOM team's best result was ranked in fifth place and among all the participating groups we were second.

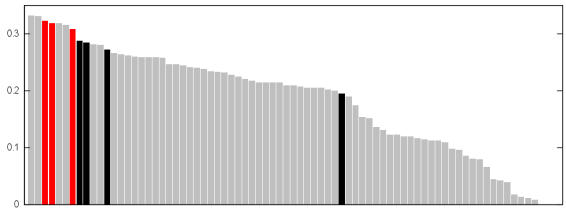

Figure 11. Results of the TRECVID 2014 semantic indexing task.

The latest developments in the PicSOM back end's performance in TRECVID 2014 SIN are shown by the red bars in Figure 11. In our best original submission, we used two deep convolutional neural networks [33] and [75] for feature extraction and obtained an MXIAP result of 0.228. In this paper, by including the DCNN features generated with [59] and [56], we improved the MXIAP to 0.308. Further improvements

were obtained by implementing a learning mechanism for the feature weights. With concept independent weighting, our best precision was 0.319, and when the feature weights were tuned for each concept separately, we scored an MXIAP of 0.323, ranking third in the TRECVID 2015 SIN task. This result shows that the performance of the PicSOM back end is on a par with the state-of-the art in video shot keyword annotation.

Figure 11 illustrates the MXIAP results of the TRECVID 2014 SIN. The black bars denote the PicSOM back end's original submitted results; the red bars (third, fourth and seventh bar on the left-hand side) show the results with the current implementation in the VisualLabel framework, whereas the gray bars are other teams' submissions in TRECVID 2014.

4.3.2. Video Shot Caption Generation

We tested the video caption generation system of the PicSOM back end by participating in the Large Scale Movie Description Challenge 2015. This was the first time the challenge was organized and only four groups out of 60 who had registered for the challenge were able to submit their results. The task was to generate captions to describe the visual content of video clips extracted from 17 videos in the public test set and 20 videos in the blind test set. In both cases, there were approximately 10 000 clips with an average duration of 5 seconds per caption.

The generated captions were evaluated both by automatic measures and by humans. In the automatic evaluation, the PicSOM system was ranked third in the challenge. However, when human evaluators were asked to assess the correctness, grammar, relevance, and helpfulness for a random sample of 1200 captions generated by the participating systems, our back end scored best with respect to the first three measures and second with respect to the last one. We can thus conclude that the PicSOM back end of VisualLabel represents the state-of-the art in sentence-based video content description and therefore we have used it for image label caption generation.

4.4. Summary

In this paper we presented novel integrated open source multimedia content management and access framework, called VisualLabel. The framework enables smart photo services based on automated visual content analysis, annotation, search and retrieval using the start-of-the-art analysis back ends for social media services, such as Facebook and Flicker. The framework includes REST APIs designed for information exchange between clients, front end, back ends, and social media and network-drive services. Furthermore, a proof-of-concept implementation of the framework is available as an open source solution [64].

The user's visual content is sent to the back ends for analysis, and based on the results different views of the same content can be presented to the user. The user can modify, add, or delete the metadata generated by the back-ends, and feedback can be provided to the back ends for further learning and improvement of content analysis. Back-end services use state-of-the-art deep learning methods for content analysis and provide the front end with the results. The text summarizer back end uses text from social media sites, processes it, and combines it with CMUVIS and PicSOM generated labels for the purposes of data visualization.

Based on the performed user study, it is concluded that the service presented here can be helpful for visual content management and organization needs.

References

[1] I. Ahmad, M. Gabbouj, A Generic Content-based Image Retrieval Framework for Mobile Devices. Journal of Multimedia Tools and Applications, Springer Science+Business Media, LLC 2010 (2010).

[2] T. Aihkisalo, T. Paaso, Latencies of Service Invocation and Processing of the REST and SOAP Web Service Interfaces. In proceedings of IEEE Eighth World Congress on Services (SERVICES), Honolulu, Hawaii, USA (2012), 100-107.

[3] M. Aqdas, M. Nieminen, Understanding the Usage and Requirements of the Photo Tagging System. Human IT, Volume 12, Issue 3 (2014), 117-161.

[4] S. Ardizzoni, I. Bartolini, M. Patella, Windsurf: Region-based image retrieval using wavelets. In Proceedings of Tenth International Workshop on Database and Expert Systems Applications (1999), 167-173.

[5] I. Bartolini, P. Ciaccia, M. Patella, Adaptively browsing image databases with PIBE, Multimedia Tools and Applications, Volume 31, Issue 3 (2006), 269-286.

[6] Baidu, http://www.baidu.com Accessed 23 February 2016.

[7] Charles Bihis, Mastering OAuth 2.0, Packt Publishing (2015).

[8] D. Cai, X. He, Z. Li, W. Y. Ma, J. R. Wen, Hierarchical clustering of www image search results using visual, textual and link information. In proceedings of the ACM International Conference on Multimedia (2004).

[9] S. A. Chatzichristofis, K. Zagoris, Y. S. Boutalis, N.Papamarkos, Accurate image retrieval based on compact composite descriptors and relevance feedback information, International Journal of Pattern Recognition and Artificial Intelligence (2009).

[10] G. Ciocca, I. Gagliardi, R. Schettini, Quicklook 2: An integrated multimedia system. Journal of Visual Languages & Computing, Volume 12, Issue 1 (2001), 81-103.

[11] Describing and Understanding Video & The Large Scale Movie Description Challenge (LSMDC), at ICCV 2015, https://sites.google.com/site/describingmovies/challenge Accessed 23 February 2016.

[12] R. Datta, Dhiraj Joshi, Jia Li, James Z. Wang, Image Retrieval: Ideas, Influences, and Trends of the New Age. ACM Computing Surveys, Volulme 40, Issue 2, 1-60 (2000).

[13] FaceBook, https://www.facebook.com/facebook/info/?tab=page_info Accessed 23 February 2016.

[14] Facebook, Manually Build a Login Flow, https://developers.facebook.com/docs/facebook-login/manually-build-a-login-flow Accessed 23 February 2016.

[15] R.E. Fan, K.W. Chang, C.J. Hsieh, X.R. Wang, C.J. Lin, LIBLINEAR: A library for large linear classification. Journal of Machine Learning Research 9 (2008), 1871-1874.

[16] X. Feng, J. Shen, Y. Fan, REST: An Alternative to RPC for Web Services Architecture. In proceedings of the First International Conference on Future Information Networks (ICFIN), Beijing, China (2009).

[17] R.T. Fielding, Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, USA (2000).

[18] R.T. Fielding, N. Richard, Principled design of the modern Web architecture. ACM Transactions on Internet Technology, Volume 2, Issue 2 (2002), 115-150.

[19] Flickr, https://www.flickr.com/about Accessed 23 February 2016.

[20] B. Gao, T. Y. Liu, T. Qin, X. Zheng, Q. S. Cheng, W. Y. Ma, Web image clustering by consistent utilization of visual features and surrounding texts. In proceedings of the ACM International Conference on Multimedia (2005).

[21] Google, https://www.google.com Accessed 23 February 2016.

[22] Google Identify Platform, Using OAuth 2.0 to Access Google APIs, https://developers.google.com/identity/protocols/OAuth2 Accessed 23 February 2016.

[23] D. Guinard, I. Ion, S. Mayer, In Search of an Internet of Things Service Architecture: REST or WS-*? A Developers' Perspective, Mobile and Ubiquitous Systems: Computing, Networking and Services, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Volume 104, Springer Berlin Heidelberg (2012), 326-337.

[24] S. Hochreiter, J. Schmidhuber, Long short-term memory. Neural Computation, Volume 9, Issue 8, (1997), 1735-1780.

[25] imprezzeo, http://www.imprezzeo.com/about/overview/ Accessed 23 February 2016.

[26] S. Ishikawa, M. Koskela, M. Sjöberg, R.M. Anwer, J. Laaksonen, E. Oja, PicSOM experiments in TRECVID 2014. In proceedings of the TRECVID 2014 Workshop. Orlando, FL, USA (2014).

[27] JQuery, https://jquery.com Accessed 23 February 2016.
[28] Y. Kalantidis, G. Tolias, Y. Avrithis, M. Phinikettos, E. Spyrou, P. Mylonas, S. Kollias, VIRaL: Visual Image Retrieval and Localization. In Multimedia Tools and Applications, Springer, Volume 51, Number 2 (2011), 555-592.
[29] A. Karpathy, L. Fei-Fei, Deep visual-semantic alignments for generating image descriptions. In proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015).
[30] S. Kiranyaz, S. Uhlmann, J. Pulkkinen, M. Gabbouj, and T. Ince, Collective network of evolutionary binary classifiers for image retrieval, in Proceedings of the IEEE Workshop on Evolving and Adaptive Intelligent Systems, EAIS 2011, Paris, France (2011), 147-154.
[31] S. Kiranyaz, J. Pulkkinen, T. Ince and M. Gabbouj, Multi-dimensional Evolutionary Feature Synthesis for Content-based Image Retrieval, in Proceedings of the IEEE International Conference on Image Processing, ICIP 2011, Brussels, Belgium (2011), 3645 - 3648.
[32] M. Koskela, J. Laaksonen, Convolutional network features for scene recognition. In proceedings of the 22nd ACM International Conference on Multimedia, Orlando, Florida (2014).
[33] A. Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, ImageNet classification with deep convolutional neural networks. In Advances in neural information processing systems (2012), 1097–1105.
[34] M. O. Lam, T. Disney, D. S. Raicu, J. Furst, D. S. Channin, BRISC: An Open Source Pulmonary Nodule Image Retrieval Framework. Journal of Digital Imaging. Vol 20, Supplement 1 (2007), 63-71.
[35] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, C. L. Zitnick, Microsoft COCO: Common objects in context. In European Conference on Computer Vision (ECCV) (2014).
[36] D. G. Lowe, Object recognition from local scale-invariant features. In ICCV, (1999) , 1150–1157.
[37] H. Luan, Y.T. Zheng, M. Wang, T.S. Chua, Visiongo: towards video retrieval with joint exploration of human and computer. Inform. Sci. 181 (2011), 4197–4213.
[38] MS COCO Captioning Challenge, http://mscoco.org/dataset/#captions-challenge2015 Accessed 23 February 2016.
[39] MUVIS, http://muvis.cs.tut.fi Accessed 23 February 2016.
[40] OneDrive, https://onedrive.live.com Accessed 23 February 2016.
[41] P. Over, G. Awad, M. Michel, J. Fiscus, W. Kraaij, A.F. Smeaton, G. Quénot, R. Ordelman, TRECVID 2015 -- an overview of the goals, tasks, data, evaluation mechanisms and metrics. In proceedings of TRECVID 2015. NIST, USA, (2015).
[42] Picasa, https://picasa.google.com Accessed 23 February 2016.
[43] T. Quack, U. Monich, L. Thiele, B. S. Manjunath, Cortina: A system for largescale, content-based Web image retrieval. In proceedings of the ACM International Conference on Multimedia (2004).
[44] P. Rantanen, P. Sillberg, J. Soini, Content Analysis System for Images. In proceedings of the 16th International Multiconference Information Society (IS 2013), Volume A, 7-11, Josef Stefan Institut, Ljubljana, Slovenia (2013), 241-244.
[45] J. Raitoharju, H. Zhang, E.C. Ozan, M.A. Waris, M. Faisal, G. Cao, M. Roininen, I. Ahmad, R. Shetty, S. P.C., S. Uhlmann, K. Samiee, S. Kiranyaz, M. Gabbouj, TUT MUVIS Image Retrieval System Proposal For MSR-Bing Challenge 2014. In proceedings of IEEE International Conference on Multimedia & Expo, ICME 2014, Chengdu, China (2014).
[46] retrievr, http://labs.systemone.at/retrievr/about Accessed 23 February 2016.
[47] L. Richardson, S. Ruby, RESTful Web Services, O'Reilly Media (2007).
[48] C. Riva, M. Laitkorpi, Designing Web-Based Mobile Services with REST. Service-Oriented Computing - ICSOC 2007 Workshops, Lecture Notes in Computer Science, Volume 4907 (2009), 439-450.
[49] A. Rohrbach, M. Rohrbach, N. Tandon, B. Schiele, A dataset for movie description. In proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015).
[50] Y. Rui, T. Huang, S. Mehrotra, Content-based image retrieval with relevance feedback in Mars. In Proceedings of the IEEE International Conference on Image Processing (ICIP) (1997).
[51] S. Sagiroglu, D. Sinanc, Big data: A review. In Proceedings of the IEEE International Conference on Collaboration Technologies and Systems (CTS) (2013), 42-47.
[52] P. Sillberg, P. Rantanen, J. Soini, A Content Based Tool For Searching, Connecting and Combining Digital Information - Case: Smart Photo Service. In Proceedings of the 16th International Multiconference Information Society (IS 2013), Volume A, 7-11, Josef Stefan Institut, Ljubljana, Slovenia (2013), 249-252.
[53] K.E.A. van de Sande, T. Gevers, C.G.M. Snoek, Evaluating color descriptors for object and scene recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 32, Issue 9 (2010), 1582-1596.
[54] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, Y. LeCun, OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. In proceedings of ICLR (2014).

[55] R. Shetty, J. Laaksonen, Video captioning with recurrent networks based on frame- and video-level features and visual content classification, ICCV 2015 Workshop on Describing and Understanding Video & The Large Scale Movie Description Challenge, Santiago, Chile (2015).
[56] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition ArXiv: 1409.1556 (2014).
[57] J. R. Smith, S. F. Chang, VisualSeek: A fully automated content based image query system. In proceedings of ACM International Conference on Multimedia, Boston, MA, USA (1996), 87–98.
[58] J. R. Smith, S. F. Chang, Visually searching the web for content. IEEE Multimedia Magazine, Volume 4, Issue 3 (1997), 12-20.
[59] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich (2014) Going deeper with convolutions. ArXiv: 1409.4842.
[60] A. Torabi, P. Chris, L. Hugo, C. Aaron, Using descriptive video services to create a large data source for video annotation research. ArXiv: 1503.01070 (2015).
[61] Twitter, https://about.twitter.com, accessed 23 February 2016.
[62] Vander Wal, Folksonomy Definition and Wikipedia, http://www.vanderwal.net/random/entrysel.php?blog=1750, accessed 23 February 2016.
[63] O. Vinyals, A. Toshev, S. Bengio, D. Erhan, Show and tell: A neural image caption generator. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015).
[64] VisualLabel source code repository, https://github.com/visuallabel, accessed 23 February 2016.
[65] Ji Wan, Dayong Wang, Steven Chu Hong Hoi, Pengcheng Wu, Jianke Zhu, Yongdong Zhang, Jintao Li, Deep learning for content-based image retrieval: A comprehensive study. In proceedings of the ACM International Conference on Multimedia (2014), 157–166.
[66] H. Wang, A. Kläser, C. Schmid, C. Liu, Dense trajectories and motion boundary descriptors for action recognition. International Journal of Computer Vision, Volume 103, Issue 1 (2013), 60-79.
[67] J. Z. Wang, J. Li, G. Wiederhold, SIMPLIcity: Semantics-Sensitive Integrated Matching for Picture Libraries. IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 23, Issue 9, (2001), 947-963.
[68] X. J. Wang, W. Y. Ma, Q. C. He, X. Li, Grouping Web image search result. In proceedings of the ACM International Conference on Multimedia (2004).
[69] L. Wu, S. C. H. Hoi, Enhancing bag-of-words models with semantics-preserving metric learning. IEEE MultiMedia, Volume 18, Issue 1 (2011), 24–37.
[70] J. Wu and J. M. Rehg, Centrist: A visual descriptor for scene categorization. IEEE Trans. Pattern Anal. Mach. Intell.,Volume 33, Issue 8 (2011), 1489–1501.
[71] Yahoo, https://www.yahoo.com Accessed 23 February 2016.
[72] Yandex Translate API, https://tech.yandex.com/translate Accessed 23 February 2016.
[73] J. Yang, Y. G. Jiang, A. G. Hauptmann, C. W. Ngo, Evaluating bag-of-visual-words representations in scene classification. Multimedia Information Retrieval (2007), 197–206.
[74] Youtube, https://www.youtube.com/yt/about/en-GB Accessed 23 February 2016.
[75] M. Zeiler, R. Fergus, Visualizing and understanding convolutional networks. ArXiv: 1311.2901 (2013).
[76] E. Zavesky, S. F. Chang, Cuzero: Embracing the frontier of interactive visual search for informed users. In ACM MIR (2008).
[77] H. Zhang, S. Kiranyaz, M. Gabbouj, Cardinal sparse partial least square feature selection and its application in face recognition. 22nd European Signal Processing Conference, EUSIPCO 2014, Lisbon, Portugal (2014).
[78] H. J. Zhang, C. Y. Low, S. W. Smoliar, J. H. Wu, Video parsing retrieval and browsing: An integrated and content-based solution. In proceedings of ACM Multimedia, San Francisco, California, USA (1995), 15-24.

Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O.B. 527
FI-33101 Tampere, Finland

ISBN 978-952-15-3912-1
ISSN 1459-2045