Technische Universität Berlin

Institut für Telekommunikationssysteme
Fachgebiet Architektur der Vermittlungsknoten

Fakultät IV
Franklinstrasse 28-29
10587 Berlin
http://www.av.tu-berlin.de

Diploma Thesis Proposal

Contextual and Conditional Content Distribution

Abdulrahman Hammood

Matriculation Number: 227675
01.01.2050

Supervised by
Prof. Dr. Thomas Magedanz

Assistant Supervisor
Dr. Adel Al-Hezmi

Contents

List of Figures

1 Introduction

The Internet has become an essential part of our daily lives in different sectors business, social communications, healthcare, etc. It has revolutionized our economy and society and being therefore considered at the top of the technological revolution in the current century. The success of the Internet can be seen on its traffic growth. The monthly global Internet traffic is expected to quadruple between 2010 and 2015 growing up from about 20.2 exabytes to 80.5 exabytes (one exabyte equals one billion gigabytes)[4]. Indeed, this growth indicates how huge the content is (and being increased rapidly) that is uploaded and consumed. Around one million minutes of video content will cross the network per second in 2015 [4]. Around one hour You-tube videos are being uploaded per second and more than four billion views per day[21]. The video-sharing content in You-tube is only one example of a huge number of distributed contents on the Internet provided by various content delivery platforms. These platforms provide different types of contents, i.e. contents are heterogeneous and can be anything, e.g. video, multimedia, books, etc.

The recent growth of multimedia content offered by multiple professional content providers (e.g. IPTV or mobile TV provider), available in several multimedia-based social networking communities or distributed in various user devices seems to be clear evidence for the need of an efficient multimedia provisioning framework that supports efficient and personalized provisioning and discovery mechanisms of multimedia content information comparing to the classical client-server provisioning systems. This thesis will address arise from the wealth of distributed multimedia content either in any controllable network or in user private network. The challenge is to provide users with technical means for rapid and instant access to relevant, trustworthy multimedia content information and enriched personalization.

Nowadays devices (e.g. PCs, smartphones, positioning devices, health monitors) in our environment are expected to work on high levels of independence, performing programmed actions that benefit their users in everyday life.In order to meet the set of requirements, these different devices are connected together performing certain tasks. This concept is known as Machine-to-Machine (M2M) communication. M2M is a concept that defines the rules and relations between devices while cooperating. It implies a highly automated usage of a set of devices simultaneously, without much need for human interaction. Although with the in-

crease of computational power, now it is even possible to run different M2M tasks on various consumer electronics (e.g. television sets, set-top boxes), smartphones are still more frequent used in M2M domain.

The aim of this thesis is to develop a context-aware platform in which context information of multimedia content environment (such as location and time) is considered and embedded into the captured multimedia content as content information or metadata. The proposed platform uses M2M concept, which provides several resources such as sensor informations like temperature, GPS data, and so on. These resources can be used to enrich the metadata of the captured content. The motivation behind this work is due to the lack of context/content aware storage management in the current Internet architecture which is one of its fundamental limitations [9]. Consumer, who is interested in certain multimedia content service, submits the request with defined conditions (e.g. time, location, content provider, etc.) that are evaluated against multimedia context information in order to deliver the associated multimedia content information (e.g. content resources, description, etc.). User-specific conditions can be published once or updated regularly. The multimedia content service shall examine user-specific conditions and notifies the user with matched multimedia content. Therefore, an efficient interactions model between consumer and the service will be developed. Furthermore, an end-to-end multimedia content service will be implemented in order to demonstrate the developed concept.

1.1 Motivation

Context-aware systems can help people in many areas of daily life to plan the daily schedule, to make important decisions correctly and perform other tasks instead of user.

Due to the fact that increased computing power of today's mobile devices, they perform tasks that were still a few years ago not possible. For these devices, with their increasingly complex applications, context-aware behaviour is of importance. Reliable and easy-used context-aware systems are required because of the explosive growth of content consumption from mobile and social interfaces and the consumer expectation of content availability. According to a statistic result reported by The Nielsen Company, U.S mobile video viewers have grown from 23 millions in the third quarter of 2010 to 31 millions in the third quarter of 2011.

M2M technology is growing fast and in the near future it will be available everywhere. Such a technology will help to enrich context information associated with multimedia content.

There are several reasons for working on this diploma thesis. On one hand, there is no correlation between contents and contexts nowadays supported by the

current Internet architecture. However, there is a demand for solutions or products that simplify the usage of the distributed content based on its contexts. The unavailability of such solutions or products is one of the main motivations behind this thesis. Within the context of this thesis, a new valid prototyping for context-aware content management platform will be developed. Therefore this thesis can set the ground for future investigation and can further be used as a cornerstone and give directions for design of better and generally accepted.

On the other hand, working on this thesis gives me the chance to get in-depth knowledge and hands-on experience of a hot topic that will evolve, improve, develop in the years to come and eventually will become inevitable part of normal way of living.

1.2 Objective

The main objective of this thesis is to develop a contextual content management platform in which the context information (metadata) is created automatically during content capturing and relying on local and distributed sensing information. The platform will support the correlation and synchronization of context information and multimedia content stream. To deliver the content to various devices the distribution mechanism will be implemented. The distribution mechanism has to be aware on device properties i.e screen resolution or internet speed. For context enrichment, the platform will use the benefits of M2M concept.

Figure 1.1 shows the main component of the platform:

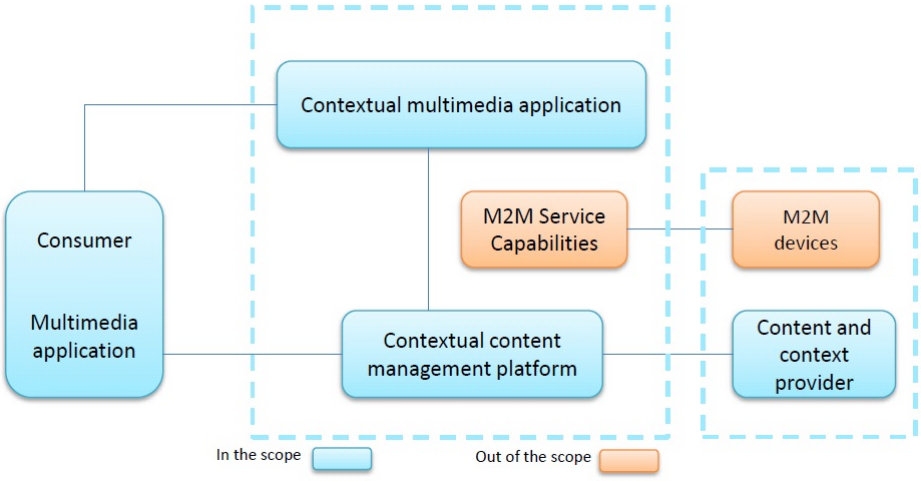

Figure 1.1: Overall Architecture

1.3 Scope

Due to lack of time and the possible wide range of technologies which have been mentioned above, the objective will be defined here in details to decide what should be developed in this thesis.

The scope of this thesis is to manage the relationship between content and context. For managing this relationship the thesis will study the state of the art of data model description and then choose the appreciate one. The thesis will also integrate M2M concept for context enrichment.

The activities in this thesis are outlined as follows:

- Act. 1: Study the state of the art of data model format, context description language
- Act. 2: Defining concept for discovering sensors (locally deployed or with M2M platform), subscriptions and notifications for sensor information.
- Act. 3: Investigate the process of automatic creation of context information and data fusion (correlation between context and content)
- Act. 4: Design the required management and delivery platforms
- Act. 5: Examine the available open sources for content management systems and HTTP-based streaming servers
- Act. 6: Based on available open source solutions, develop an end-to-end platform that enables content provider(mobile app.) to capture multimedia content with the associated context information and publish this content to the server and allows consumers(mobile app. or web based) to discover and subscribe to multimedia content according to defined conditions.
- Act. 7: Develop an end-to-end application to evaluate the implemented functionality
- Act. 8: Validate the implementation through an end-to-end deployment scenario that is planned to be deployed in the FOKUS open EPC(Evolved Packet Core) or MTC(Machine Type Communication) playground. In the first deployment scenario, the quality of content streaming to the consumer or network selection will be instantly identified according to user and network policies. In the second deployment scenario, content-related context information – in particular geographical and time information of multimedia content stream – can be considered as collaborative crowd sourcing with multimedia content that can enrich any MTC platform.

Figure 1.2 shows inputs and outputs for this thesis:

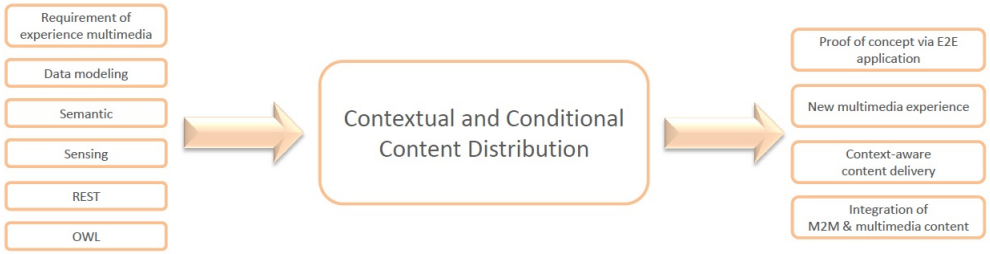

Figure 1.2: Inputs & Outputs

By studying these tasks the goal is to draw conclusions about best practices in this domain, and design the platform that can establish the basis for its further development and future implementation.

1.4 Methodologie

1.5 Outline

2 Background and Related Work

dkjekdede dejfekjfhkjehf jefkjefkjefkje jekfnkejfbkjebfkje kejfbnekjfbjkebfekjfnekjfbnkjebf kjenfk jbkejfkjenfkj nejk kjn kj kjbjkbkjbkjb kj bkj jk bkjb kjb kj bkj bkj bkj bkjb kj kjbkjbkjbkjbkj kjhkjhkjhjk kjhkjhjkh kjhkjhkjh kjhkjhk kjhjkhkj.

2.1 Context-awareness

In this section, the terms "context" and "context-aware computing" are discussed in more detail. Furthermore, application possibilities of context awareness is demonstrated.

In order for computers to assist users in their everyday tasks, they should adapt themselves to the current user's situation, and then respond according to this situation.

Humans are quite successful at conveying ideas to each other and reacting appropriately. This is due to many factors: the richness of the language they use, the common understanding of how the world works, and an implicit understanding of every-day situations. When humans talk with each other, they are able to use implicit situational information, or *context*, to increase the conversational bandwidth. Unfortunately, this ability to convey ideas does not transfer well to humans interacting with computers.Dey and Abowd

2.1.1 What is Context?

The report that first introduces the term *context-aware*, [Schilit and Theimer] refers to context as location, identities of people and objects nearby, and changes to those objects. A similar definition, [Brown et al.] describes context as location, identities of the people around the user, the time of day, season, temperature, etc. [Ryan et al.] defines context as the user's location, environment, identity and time. [Dey] enumerates context as the user's emotional state, focus of attention, location and orientation, date and time, objects, and people in the user's environment.

A recent definition of context-awareness is described by [Dey and Abowd] who defines it as, Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered

relevant to the interaction between a user and an application, including the user and applications themselves.

This way it is easier for an application developer to enumerate the context for a given application scenario. If a piece of information can be used to characterize the situation of a participant in an interaction, then that information is regarded as context.

After the term "context" has been defined the term "context-aware computing" is discussed in the following subsection.

2.1.2 Context-aware Computing

Context-aware computing was first described by [Schilit and Theimer] in 1994 to be software, that adapts according to its location of use, the collection of nearby people and objects, as well as changes to those objects over time. However, it is commonly agreed that context-aware computing was first investigated in 1992 by [Want et al.].

[Ryan] has also defined the term context-aware applications as applications that allow users to select from a range of physical and logical contexts according to their current interests or activities and also monitor input from environmental sensors.

2.1.3 Context Examples

The orientation of the screen of a tablet computer is automatically changed, maps can orientate themselves according to the user's direction with the zoom level adapted to the current speed, and the backlight of the phone is switched on when used in the dark.

These are examples of computers that are aware of their environment and their contextual use. However such functions were not common 10 years ago and only existed on prototype devices in research labs which researched context-aware computing.

Below are also some examples for context awareness in mobile and non-mobile environments. Although non-mobile environments for this thesis are not relevant, they are interesting at this point in order to show the diverse application areas which illustrate the usage Context-Awareness systems.

- identity
- spatial information
  e.g. location, orientation, speed, and acceleration

- temporal information
  e.g. time of the day, date, and season of the year
- environmental information
  e.g. temperature, air quality, and light or noise level
- social situation
  e.g. who you are with, and people nearby
- resources that are nearby
  e.g. accessible devices, and hosts
- availability of resources
  e.g. battery, display, network, and bandwidth
- physiological measurements
  e.g. blood pressure, heart rate,respiration rate,muscle activity, and tone of voice
- activity
  e.g. talking, reading, walking, and running

2.2 Context Description

In order to efficiently use the context data after acquisition, it needs to be represented and/or stored in an appropriate form suitable for further processing. Now some of the different types of context modeling will be discussed.

- **Key-value model:** This modeling technique represents contextual information with key-value pairs which is one of the most simple data structures for modeling contextual information. This model was already used in 1994 by Schilit et al. to present the context by providing the value of a context information (e.g. location information) to an application as an environment variable. Distributed service frameworks frequently use the key-value modeling approach. Although key-value pairs lack capabilities for sophisticated structuring for enabling efficient context retrieval algorithms, they are easy to manage.
- **Logic based model:** Logic-based models have a high degree of formality. Typically, facts, expressions and rules are used to define a context model. [2]

  A logic defines the conditions on which a concluding expression or fact may be derived (a process known as reasoning or inferencing) from a set of other

expressions or facts. To describe these conditions in a set of rules a formal system is applied. In a logic based context model, the context is consequently defined as facts, expressions and rules. Usually contextual information is added to, updated in and deleted from a logic based system in terms of facts or inferred from the rules in the system respectively. Common to all logic based models is a high degree of formality.[18]

In early 1993 Mccarthy and his group at Stanford researched one of the first logic based context modeling approaches and published it as a "Notes on formalizing contexts". They introduced contexts such as abstract mathematical entities with properties useful in artificial intelligence.

- **Ontology based model:** Ontologies are a promising instrument to specify concepts and interrelations [10]. The Web Onotlogy Language (OWL) is one way of implementing these ontologies. This consists of a set of classes, class hierarchies, set of property assertions, constraints on these elements, and types of permitted relationships between them. Another way to implement the ontologies is to use a knowledge representation language - the Resource Description Framework (RDF). This is a promising model because of the possibility to apply reasoning techniques [12].

  Öztürk and Aamodt proposed one of the first approaches of modeling the context with ontologies. Psychological studies on the difference between recall and recognition of several issues in combination with contextual information were analyzed by them. The necessity of normalizing and combining the knowledge from different domains was derived from this examination. A context model based on ontologies due to their strengths in the field of normalization and formality was proposed by them.

- **Graphical models:** The Unified Modeling Language (UML) is a wide spread modeling tool for software systems. When using UML, the architectural aspects of software systems are defined as classes. Each class constitutes a set of objects with common services, properties, and behavior. Services are described by methods and properties are described by attributes and associations [17].

- **Object-oriented models:** Object-oriented design of context benefits from the common properties object-oriented programming, such as inheritance, encapsulation, reuse, and polymorphism. As an example, in a model of a Payroll System, a Company is an Object. An Employee is another Object. Employment is a Relationship or Association. An Employee Class (or Object for simplicity) has Attributes like Name, Birthdate, etc. The Association

itself may be considered as an Object, having Attributes, or Qualifiers like Position, etc. An Employee Method may be Promote, Raise, etc [20].

- **Markup languages:** All markup scheme modeling approaches share a hierarchical data structure consisting of markup tags with attributes and content. In particular, the content of the markup tags is usually recursively defined by other markup tags. Typical representatives of this kind of context modeling approach are profiles. They usually base upon a serialization of a derivative of Standard Generic Markup Language(SGML), the superclass of all markup languages such as the popular XML[18].

2.3 Related Technologies

This section covers a wide of technologies which are relevant for this thesis.

2.3.1 Web Services

There are many definitions for the term Web service, the World Wide Web Consortium (W3C) defines it as follows:[1]

> A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

The W3C also states:[1]

> We can identify two major classes of Web services, REST-compliant Web services, in which the primary purpose of the service is to manipulate XML representations of Web resources using a uniform set of "stateless" operations; and arbitrary Web services, in which the service may expose an arbitrary set of operations.

By using Web Services, it is now very easy to make existing data and functions from existing applications available to consumers. Web services are considered as a "machine to machine" communication, which exchange messages via standard protocols.

The most known web services technologies TODO akronom SOAP and TODO REST are discussed in the following sections.

SOAP

SOAP stands for "Simple Object Access Protocol" and it relies on TODO akro. XML for its message format. For message negotiation and transmission, it is dependent on other Application Layer protocols, particularly Hypertext Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP), however HTTP has gained wider acceptance as it works well with today's Internet infrastructure and also with network firewalls.

A SOAP message is a type of envelope or container, which may contain an optional header element and a mandatory body element, see figure 2.1. Meta-data for this message are located in the header and the user data are stored in the body.

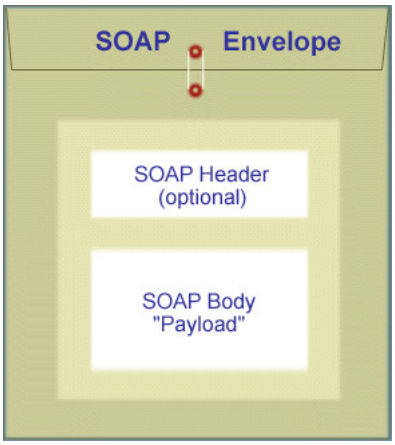

Figure 2.1: SOAP Envelop

SOAP Message Example: The following example gives an overview of a SOAP message:

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock">
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

REST

In addition to SOAP, there is another alternative for the implementation of Web services. Fielding in his dissertation describes an architectural style that he calls REpresentational State Transfer architecture or short REST.

REST is based on principles that are used in the largest distributed application - the World Wide Web. The World Wide Web is itself a gigantic REST application. Many search engines, shops or booking systems have unintentionally been based on REST web services.

The REpresentational State Transfer Architecture is an architectural model, which describes how the Web should work. The model will serve as a guide and reference for future enhancements.

REST is not a product or standard. REST describes how web standards in a Web-friendly manner can be used.

REST Example: An online store will serve here as an example of a RESTful application. In this application, there are customers who can place items in shopping carts.

Each object of the application, such as the product or the customer is a resource that is externally accessible via a URL. With the following request in the example application, the shopping cart with the number 7621 is retrieved.

```
GET /cart/7621
```

It is not specified in REST how the result of a request is represented. Client and server must have a shared understanding how the data is represented, i.e. in XML or JSON. The following example is a response in JSON format.

```
{
"customer": 7621,
"articles":[
        {"position":1,
        "articleNumber"=89,
        "description":"iPhone5",
        "price":200},
        {"position":2,
        "articleNumber"=76,
        "description":"Samsung Galaxy S III",
        "price":150}
]
}
```

SOAP vs. REST TODO

The main advantages of REST web services is that they are lightweight, without a lot of extra XML markup. Also REST has easy to read results and is easy to build requiring no special tool-kits.

SOAP also has some advantages, usually it is easy to use, provides relatively strong typing since it has a fixed set of supported data types, furthermore many different kinds of development tools are available.

Next some aspects of SOAP and REST will be compared.

API Flexibility & Simplicity The key to the REST methodology is to use an interface that is already well known and widely used, the URI, in order to write web services. For example, providing a currency converter service, in which a user types-in the desired currencies for input and output and the specific amount in order to receive a real-time conversion, could be as simple as making a script accessible on a Web server via the following URI: http://www.currencyconverter.comconvert?in=us-dollar&value=100&out=euro

This service could easily be requested with an HTTP GET command by any client or server application with HTTP support. The resulting HTTP response depends on how the service provider wrote the script and it might be as simple as some standard headers and a text string containing the current price for the given currencies, or it might be an XML document.

The significant advantages of this interface method over SOAP-based services are as follows:-

The creation and modification of a URI in order to access different web resources can easily be figured out by any developer. However, in order for SOAP to be used, most developers would need a SOAP toolkit to form requests and obtain the results, as it requires specific knowledge of a new XML specification.

Bandwidth Usage The RESTful interface has short requests and responses, which is another advantage. Whereas, an XML wrapper around every request and response is required for SOAP. For a four- or five-digit stock quote, a SOAP response may require more than 10 times the number of bytes as the same response in REST, as SOAP requires namespaces and typing to be present.

Security The security perspective debate is probably the most interesting aspect of the comparison between REST and SOAP.

Sending remote procedure calls (RPC) through standard HTTP ports is seen by the SOAP camp as being a good way to ensure Web services support across organizational boundaries. In contrast however, REST followers see this as compromising network safety and considers this practice a major design flaw.

With REST, the administrator (or firewall) can discern the intent of each message by analyzing the HTTP command used in the request, even though the REST calls also go over HTTP or HTTPS. For example, a GET request is always seen as being safe because by definition, the data cannot be modified. It can only query data.

On the other hand, HTTP POST is used by a typical SOAP request to communicate with a given service. Without looking into the SOAP envelope, it is not possible to know whether the request simply wants to query data or delete entire tables from the database. This task is resource-consuming and it is not built into most firewalls.

On the downside with SOAP, the difficult task of authentication and authorization is left up to the application developer. However, the fact that the web servers already have support for these tasks, is taken into account by the REST methodology. REST methodology developers can make the network layer do all the heavy work by using industry-standard certificates and a common identity management system, such as an LDAP server.

However, REST is not perfect. It is not always the best solution for every web service. Data should never be sent as parameters in URIs in order to be kept secures.

Type Handling Due to its fixed set of supported data types, SOAP provides a stronger typing. In this way, it ensures a return value will be given in the corresponding native type in a specific platform. For example, when an API is HTTP based, the return value will need to be deserialized from its original XML format before being type-casted. However, handling complex data-types proves to be the main challenge and is mainly achieved by defining a serialization and deserialization mechanism, wherefore there is no definitive advantage concerning ease of client-side coding.

Client-side Complexity Making calls to an REST API poses less of a challenge than making calls to a SOAP API. While REST is elementary to all programming languages and merely implies constructing an HTTP request with the appropriate parameters, the latter requires a client library, a Stub and involves additional learning effort.

Testing and Troubleshooting A further characteristic of REST APIs is their easy testing and troubleshooting ability, requiring no more than a browser, the response appearing in the browser window itself. Generating a request does not require special test tools, this is a major advantage of REST based APIs.

Server-side Complexity The majority of programming languages provide easy to operate mechanisms to expose a method using SOAP. However exposing a method using REST based APIs, involves additional effort due to the task of mapping the URI path to specific handlers. Though various frameworks facilitate this task, the exposition of methods is still easier to achieve using SOAP than REST.

Caching To consume a REST based API service, a simple GET request is needed, therewith allowing proxy servers to cache their response very easily. In contrast, SOAP requests use POST and require a complex XML format, producing difficulties for response-caching.

2.3.2 NoSQL Databases TODO NoSQL Evaluation

SQL databases have been used to solve storage problems for a long time, including cases in which there is a high discrepancy between the object model and its relational model. The conversion of graphs to tables represents yet another disfunctional use of data mapping. The complex structure this sort of mismanagement causes depends on mapping frameworks and complex algorithms. The rigid relational scheme characteristic for SQL becomes especially inefficient for such web applications as blogs due to their multifaceted range of attributes that need to be stored in their respective tables, e.g. comments, pictures, audios, videos, source codes. Therefore adding or removing a new feature to this sort of website will necessarily result in system unavailability.

Nowadays of course, web sites are developing towards more interactive models, obliging databases to perform real-time scheme updates, thereby paving the way for NoSQL to provide a database molded for modern demands.

There is a variety of ideas surrounding the NoSQL movement, however the core idea is to provide more flexible data models, as opposed to the SQL approach, in order to provide live scheme updates. The ever increasing amount of data streaming through the web implies challenges, which any competitive website wishing to stay in business will have to meet. Besides dealing with vast amounts of data, these sites have to respond to constant requests around the globe without allowing any noticeable latency.

To this end, many companies have developed their own storage systems, according to their specific needs, which have been classified as NoSQL databases.

Considering the fact that these stores are set up to fulfill the individualized requirements of the companies they belong to, there can be no final answer as to which of them works most efficiently. For example, Facebook implemented the NoSQL database Cassandra in order to solve the so called "Inbox Search Problem" - the challenge of allowing Facebook users to search through their sent and received messages - caused by the multitude of stored data alongside the high number of active users. ————————

A selection of the best known NoSQL systems are shown in Table , which are categorized into the following groups: Key Value Stores, Document Stores, Column Family Stores and Graph Databases

Key Value Stores	Document Stores	Column Stores	Graph Databases
Riak **Amazon SimpleDB** **Voldemort** **Redis**	**CouchDB** **MongoDB** **Couchbase**	**HBase** **Hypertable** **Cassandra**	**Neo4J** **AllegroGraph**

The following sections will examine these groups, each accompanied by one or more exemplary implementations.

Key Value Stores

Key value stores provide suitable storage systems for simple operations, based on key attributes only. They can be compared to maps or dictionaries due to the fact that data is identified by a unique key. They allow for a user specific webpage to be partially calculated beforehand and consequently be served quickly to the user when requested. Most key value stores save their data in memory, so they are frequently used for caching of SQL queries, which result more time intensive. Examples for key value stores are Project Voldemort, Redis,Membase and Riak of which the latter will be described in detail in the following paragraph.

Riak: Riak is a distributed, scalable, open source key/value store. Riak scales predictably and easily and simplifies development by giving users the ability to quickly prototype, test, and deploy their applications. One of two ways to access data in Riak by using a REST API. The other way to access data is through a fully-featured Protocol Buffers API. This is a simple binary protocol based on the library Google's open source project of the same name.

The only one way to organize data inside of Riak is by using buckets and keys. Data is stored and referenced by bucket/key pairs. These buckets are used to define a virtual keyspace and provide the ability to define isolated non-default

configuration. They might be compared to tables or folders in relational databases or file systems, respectively [**?**].

Interactions with Riak are mostly setting or retrieving the value of a key. The following describes how to do that using the Riak HTTP API. To read an object, this is a basic command formation for retrieving a specific key from a bucket.

```
GET /riak/bucket/key
```

The body of the response will contain the contents of the object (if it exists).

For storing an object with a user-defined key, the basic request looks like this:

```
PUT /riak/bucket/key
```

There is no need to explicitly create a bucket because they are automatically created when keys added to them.

Document Stores

Document Stores interlace key value pairs in JSON or JSON like documents. Each of these documents contains a special key "ID", which is unique throughout a collection of documents and therefore identifies a document explicitly.

Key value stores don't allow for values to be queried, because the values are only accessible through their respective keys. Document stores on the other hand provide mechanisms for this additional function. They therefore allow complex data structures to be handled more efficiently. The interpretable JSON format applied by document stores ensures a developer friendly handling by supporting data types. In contrast to key value stores which focus on high performance for read and write concurrent, document stores concentrate on storing big data efficiently while also providing a high query performance. Most known document stores are CouchDB and MongoDB which are described in detail in the following paragraphs.

CouchDB: Apache CouchDB is a free and open-source document-oriented database written in Erlang, a programming language aimed at concurrent and distributed applications. It was firstly released in 2005 and later became an Apache project in 2008.

CouchDB implements a form of Multi-Version Concurrency Control (MVCC) in order to avoid the need to lock the database file during writes. Conflicts are left to the application to resolve. Resolving a conflict generally involves first merging data into one of the documents, then deleting the stale one.

CouchDB stores data in semi-structured documents. A document is a JSON file which is a collection of named key-value pairs. Values can either be numbers, string, booleans, lists or dictionaries. The documents are completely schema-free and do not have to follow any structure (except for JSON's inherent structure).

An example document is shown in Listing 1, which represent the apple price in various supermarket.

```
{
    "_id" : "bc2a98726578c326ec68382f846d7629",
    "_rev" : "8763642898",
    "item" : "apple",
    "prices" : {
        "ALDI" : 1.59,
        "LIDL" : 5.99,
        "Kaufland" : 0.79
    }
}
```

Listing 1: Example of a CouchDB document

Each document in a CouchDB database is identified by a unique ID (the 'id' field). CouchDB is a simple container of a collection of documents and it does not establish any mandatory relationship between them [ALS10]. CouchDB exposes a RESTful HTTP API to perform basic CRUD operations on all stored items and it uses the HTTP methods POST, GET, PUT and DELETE to do so.

As illustrated in figure 2.2, one can synchronize the data between any two databases easily. After replication, each database is able to work independently.

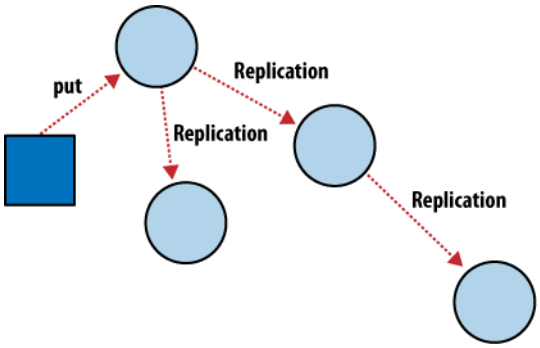

Figure 2.2: CouchDB replication

For conflict handling, CouchDB relies on the MVCC model. Each document is assigned a revision id and every time a document is updated, the old version is kept and the updated version is given a different revision id. Whenever a conflict is detected, the winning version is saved as the most recent version and the losing version is also saved in the document's history. This is done consistently throughout all the nodes so that the exact same choices are made. The application can then chose to handle the conflict by itself (ignoring one version or merging the changes) [ALS10].

MongoDB: MongoDB (from "humongous") is a free and open-source document-oriented database written in C++ [10g11d]. Aside from the open-source community, the development is also supported by the company 10gen. It is completely schema-free and manages JSON-style documents, as in CouchDB. It focuses on high-performance and agile development, providing the developer with a set of features to easily model and query data, as well as to scale the system.

MongoDB stores data as BSON objects, which is a binary-encoded serialization of JSON-like documents. It supports all the data types that are part of JSON but also defines new data types, i.e. the Date data type and the BinData type [BSO11]. The key advantage of using BSON is efficiency (both in space and compute time), as it is a binary format [10g09]. Documents are contained in 'collections', they can be seen as an equivalent to relational database tables [10g11b]. Collections can contain any kind of document, no relationship is enforced, still documents within a collection usually have the same structure as it provides a logical way to organize data. As data within a collections is usually contiguous on disk, if collections are smaller better performance is achieved [10g11h]. Each document is identified by a unique ID ('id' field), which can be given by the user upon document creating or automatically generated by the database [10g11e]. An index is automatically created on the ID field although other indexes can be manually created in order to speed up common queries.

Two forms of replication are available, Replica Sets and Master-Slave, see figure 2.3. As Master-Slave replication is deprecated since version 1.6, Replica Sets are used for all new production deployments.

Column Stores

Column Family Stores are also known as column oriented stores, extensible record stores and wide columnar stores TODO cite. All stores are inspired by Googles Bigtable, which is a "distributed storage system for managing structured data that is designed to scale to a very large size" [12] TODO cite. Google has been using Bigtable in many projects with varying requirements of high throughput and latency-sensitive data serving. The data model is described as "sparse, distributed,

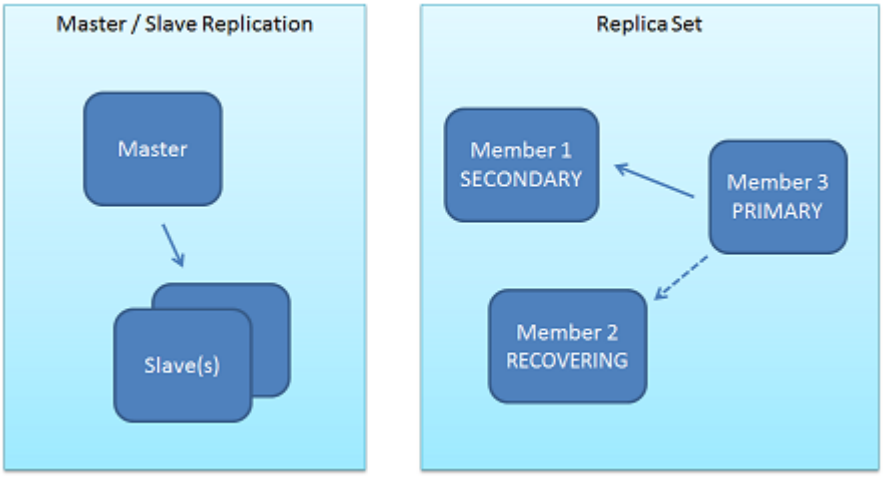

Figure 2.3: MongoDB replication

persistent multidimensional sorted map" [12] TODO cite. In this map, an arbitrary number of key value pairs can be stored within rows. Values cannot be interpreted by the system, therefore relationships between datasets and any other data types than strings are not supported natively. These additional features have to be implemented in the application logic as is also the case when handling key value stores. In order to achieve both versioning and consistency, multiple versions of a value are stored chronologically.

Big companies such as Google and Facebook, use their respective Column Family Stores (Big Table and Cassandra) to store data in the same way as it will eventually be required when requested. This sort of storage ensures that only one data retrieval is needed when a request is launched, therefore saving considerable effort and maximizing performance when compared to the conventional process via MySQL which often requires multiple queries.

Cassandra Apache Cassandra is a free and open-source distributed, structured key-value store with eventual consistency. It is a top-level project of the Apache Foundation and it was initially developed by Facebook [Fou11h]. It is designed to handle very large amounts of data, while providing high availability and scalability.

(cite Cassandra) A table in Cassandra is a distributed multi dimensional map indexed by a key. The value is an object which is highly structured. The row key in a table is a string with no size restrictions, although typically 16 to 36 bytes long. Every operation under a single row key is atomic per replica no matter how many columns are being read or written into. Columns are grouped together into sets called column fam- ilies very much similar to what happens in the Bigtable[4]

system. Cassandra exposes two kinds of columns families, Simple and Super column families. Super column families can be visualized as a column family within a column family. Furthermore, applications can specify the sort order of columns within a Super Column or Simple Column family. The system allows columns to be sorted either by time or by name. Time sorting of columns is exploited by application like Inbox Search where the results are always displayed in time sorted order. Any column within a column family is accessed using the convention column family : column and any column within a column family that is of type super is accessed using the convention column family : super column : column. A very good example of the super column family abstraction power is explained below.

For Inbox Search we maintain a per user index of all mes- sages that have been exchanged between the sender and the recipients of the message. There are two kinds of search features that are enabled today (a) term search (b) interactions - given the name of a person return all messages that the user might have ever sent or received from that person. The schema consists of two column families. For query (a) the user id is the key and the words that make up the message become the super column. Individual message identifiers of the messages that contain the word become the columns within the super column. For query (b) again the user id is the key and the recipients id's are the super columns. For each of these super columns the individual message identifiers are the columns. In order to make the searches fast Cassandra provides certain hooks for intelligent caching of data. For instance when a user clicks into the search bar an asynchronous message is sent to the Cassandra cluster to prime the buffer cache with that user's index. This way when the actual search query is executed the search results are likely to already be in memory. The system currently stores about 50+TB of data on a 150 node cluster, which is spread out between east and west coast data centers. We show some production measured numbers for read performance.

Latency Stat	Search Interactions	Term Search
Min	7.69ms	7.78ms
Median	15.69ms	18.27ms
Max	26.13ms	44.41ms

Graph Databases:

(cite NoSQL Evaluation) In contrast to relational databases and the already introduced key oriented NoSQL databases, graph databases are specialized on efficient management of heavily linked data. Therefore, applications based on data with many relationships are more suited for graph databases, since cost intensive operations like recursive joins can be replaced by efficient traversals. Neo4j [16] and GraphDB [17] are based on directed and multi relational property graphs. Nodes and edges consist of objects with embedded key value pairs. The range

of keys and values can be defined in a schema, whereby the expression of more complex constraints can be described easily. Therefore it is possible to define that a specific edge is only applicable between a certain types ofnodes. Property graphs are distinct from resource description framework stores like Sesame [18] and Bigdata [19] which are specialized on querying and analyzing subject-predicate-object statements. Since the whole set of triples can be represented as directed multi relational graph, RDF frameworks are considered as a special form of graph databases in this paper, too. In contrast to property graphs, these RDF graphs do not offer the possibility of adding additional key value pairs to edges and nodes. On the other handy, by use of RDF schema and the web ontology language it is possible to define a more complex and more expressive schema, than property graph databases do. Twitter stores many relationships between people in order to provide their tweet following service. These one-way relationships are handled within their own graph database FlockDB [20] which is optimized for very large adjacency lists, fast reads and writes. Use cases for graph databases are location based services, knowledge representation and path finding problems raised in navigation systems, recommendation systems and all other use cases which involve complex relationships. Property graph databases are more suitable for large relationships over many nodes, whereas RDF is used for certain details in a graph. FlockDB is suitable for handling simple I-hop-neighbor relationships with huge scaling requirements.

Neo4J bla bla:

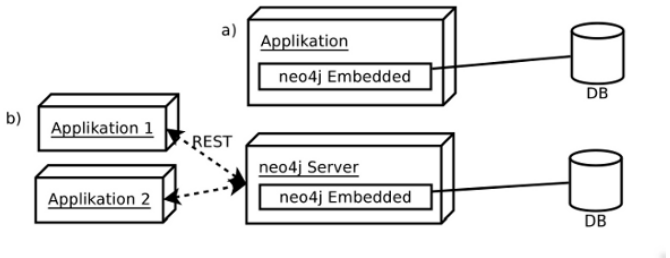

Figure 2.4: Embedded (a) vs. Server (b)

2.3.3 Message Queue

Message queues provide an asynchronous communications protocol, meaning that the sender and receiver of the message do not need to interact with the message queue at the same time. Messages placed onto the queue are stored until the

recipient retrieves them. Message queues have implicit or explicit limits on the size of data that may be transmitted in a single message and the number of messages that may remain outstanding on the queue.

There are a number of open source choices of messaging middleware systems, including JBoss Messaging, JORAM, Apache ActiveMQ, Sun Open Message Queue, Apache Qpid, RabbitMQ, Beanstalk'd, Tarantool and HTTPSQS

RabbitMQ bla bal TODO cite fig http://www.rabbitmq.com/tutorials/amqp-concepts.html

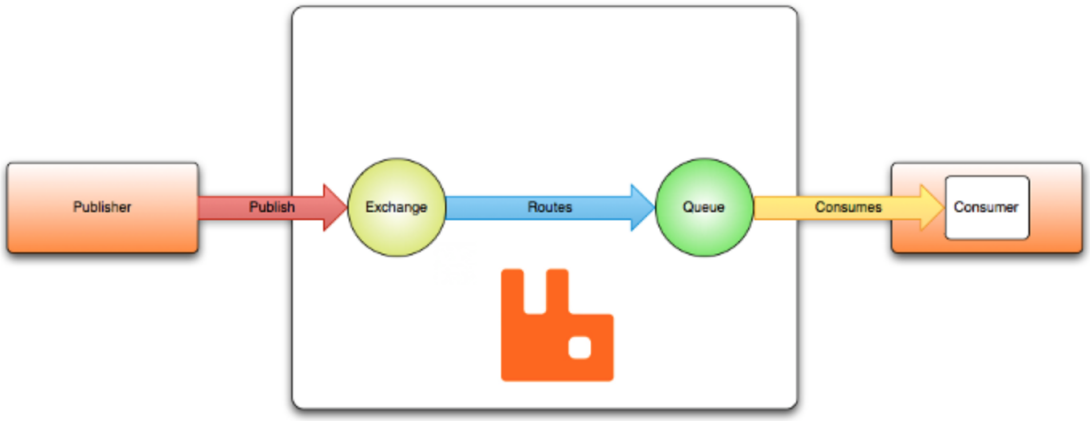

Figure 2.5: RabbitMQ Components

Direct exchange witer vom http://www.rabbitmq.com/tutorials/amqp-concepts.html bla bal TODO cite fig http://www.rabbitmq.com/tutorials/amqp-concepts.html

bla bal TODO cite fig http://www.rabbitmq.com/tutorials/amqp-concepts.html

ActiveMQ

2.3.4 Search Platforms

Search engines are designed to help users to quickly find useful information from the web. Thousands number of search engines are existing to perform the task of information retrieval but only some of them are popular. Because of vast availability of numerous search engines searchers often get confuse with the problem of good search engine selection.

User queries are becoming more complex and personalized over time, and much of the data required to deliver an appropriate response is inherently unstructured.

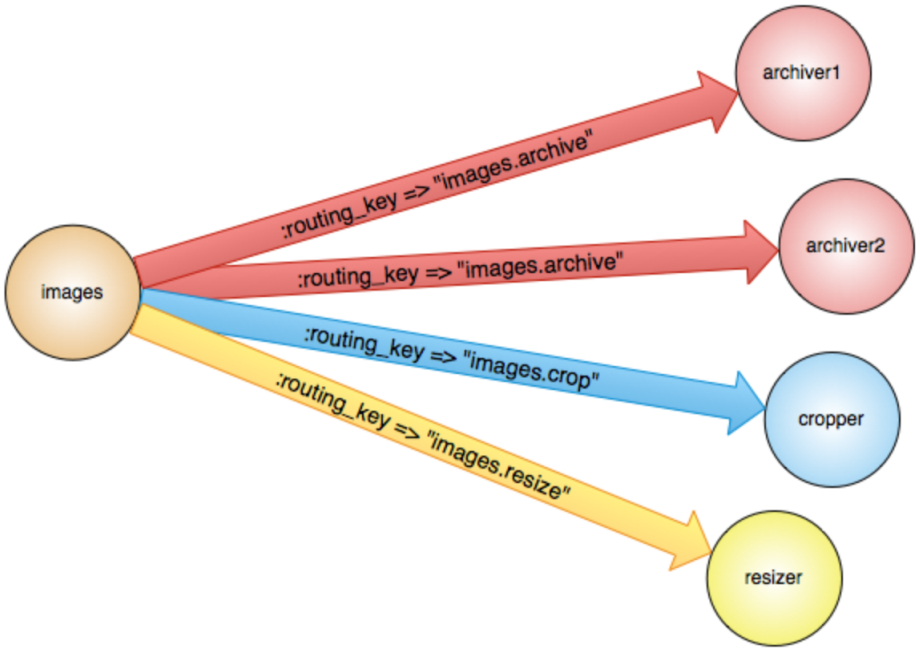

Figure 2.6: Direct exchange routing

Where once an SQL LIKE clause was good enough, today's usage sometimes calls for sophisticated algorithms. Fortunately, a number of open source and commercial platforms address the need for pluggable search technology, including Lucene, Sphinx, Solr, Amazon's CloudSearch, and Elasticsearch.

Apache Solr

cite https://svn.apache.org/repos/asf/lucene/dev/branches/lucene3622/solr/site/index.pdf

Solr is the popular, blazing fast open source enterprise search platform from the Apache Lucene project. Its major features include powerful full-text search, hit highlighting, faceted search, dynamic clustering, database integration, rich document (e.g., Word, PDF) handling, and geospatial search. Solr is highly scalable, providing distributed search and index replication, and it powers the search and navigation features of many of the world's largest internet sites. Solr is written in Java and runs as a standalone full-text search server within a servlet container such as Tomcat. Solr uses the Lucene Java search library at its core for full-text indexing and search, and has REST-like HTTP/XML and JSON APIs that make it easy to use from virtually any programming language. Solr's powerful external configuration allows it to be tailored to almost any type of application without Java coding, and it has an extensive plugin architecture when more advanced customization is required.

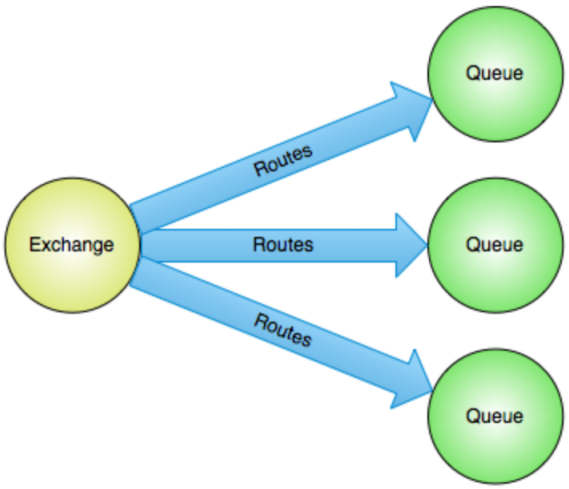

Figure 2.7: Fanout exchange routing

Elasticsearch

cite CERN:

Elasticsearch provides both an indexing service as well as a data store. It does not require a detailed schema , although one may be provided if needed. Any data that we can express as JSON can be easily stored and indexed with Elasticsearch.

Since elasticsearch is schema-less unless explicitly defined, any JSON documents can be added to elasticsearch, see following example:

```
curl -XPUT http://localhost:9200/blogs/blog/1 -d '{
    "post_date": "2012-12-15T09:00:00",
    "content": "This is the first blog"
}'

curl -XPUT http://localhost:9200/blogs/blog/2 -d '{
    "author": "abdul",
    "post_date": "2012-12-16T11:00:00",
    "content": "Second blog"
}'
```

Listing 2: schema free elasticsearch

Internally, elasticsearch uses a NoSQL database system supporting JSON documents. Due to multi tenancy, an elasticsearch instance can have more than one index. An index consists of types which consist of fields. In accordance to relational database systems, an index is equivalent to a database, a type is comparable

to a table and a field is comparable to a column. No type schema definition is necessary which makes elasticsearch highly flexible. Indexes and types are created automatically if they do not exist yet. Types such as numbers and dates are automatically detected and treated accordingly.

Elasticsearch offers so-called 'multi tenancy' allowing documents to be divided into seperate 'indexes'. An index is equivalent to a database in relational database systems. Elasticsarch offers 'sharding' allowing indexes to be broken up into smaller 'shards'. Replication for indexes and shards allows a distributed environment with copies on different nodes. Another design goal of elasticsearch is near real-time search also supported in multi tenant and multi node environments.

One of the unique features of Elasticsearch that makes it especially well suited for our purposes is the ease with which we can scale our solution. It provides us with the ability to either add or remove resources (that is, individual machines running Elasticsearch) at any time. We might do this in order to support a growing data set or, perhaps, in order to satisfy an increasing amount of requests and to improve the performance of our solution.

2.3.5 Spring Framework

cite: http://static.springsource.org/spring/docs/3.2.x/spring-framework-reference/pdf/spring-framework-reference.pdf The Spring Framework is a lightweight solution and a potential one-stop-shop for building your enterprise-ready applications. However, Spring is modular, allowing you to use only those parts that you need, without having to bring in the rest. You can use the IoC container, with Struts on top, but you can also use only the Hibernate integration code or the JDBC abstraction layer. The Spring Framework supports declarative transaction management, remote access to your logic through RMI or web services, and various options for persisting your data. It offers a full-featured MVC framework, and enables you to integrate AOP transparently into your software. Spring is designed to be non-intrusive, meaning that your domain logic code generally has no dependencies on the framework itself. In your integration layer (such as the data access layer), some dependencies on the data access technology and the Spring libraries will exist. However, it should be easy to isolate these dependencies from the rest of your code base.

Modules

The Spring Framework consists of features organized into about 20 modules. These modules are grouped into Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, and Test, as shown in the following diagram.

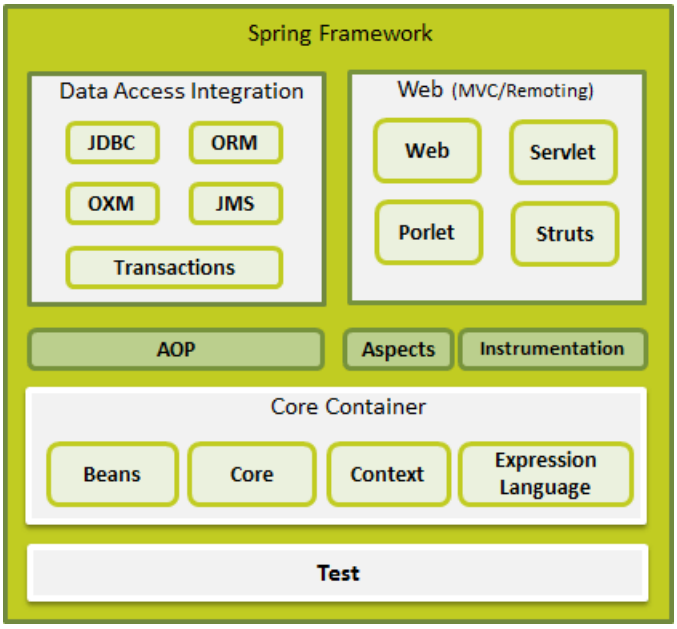

Figure 2.8: Overview of the Spring Framework

Core Container: The Core Container consists of the Core, Beans, Context, and Expression Language modules.

The Core and Beans modules provide the fundamental parts of the framework, including the IoC and Dependency Injection features. The BeanFactory is a sophisticated implementation of the factory pattern. It removes the need for programmatic singletons and allows you to decouple the configuration and specification of dependencies from your actual program logic.

The Context module builds on the solid base provided by the Core and Beans modules: it is a means to access objects in a framework-style manner that is similar to a JNDI registry. The Context module inherits its features from the Beans module and adds support for internationalization (using, for example, resource bundles), event-propagation, resource-loading, and the transparent creation of contexts by, for example, a servlet container. The Context module also supports Java EE features such as EJB, JMX ,and basic remoting. The ApplicationContext interface is the focal point of the Context module.

The Expression Language module provides a powerful expression language for querying and manipulating an object graph at runtime. It is an extension of the unified expression language (unified EL) as specified in the JSP 2.1 specification. The language supports setting and getting property values, property assignment, method invocation, accessing the context of arrays, collections and indexers, logical and arithmetic operators, named variables, and retrieval of objects by name from

Spring's IoC container. It also supports list projection and selection as well as common list aggregations.

Data Access/Integration: The Data Access/Integration layer consists of the JDBC, ORM, OXM, JMS and Transaction modules.

Web: The Web layer consists of the Web, Web-Servlet, Web-Struts, and Web-Portlet modules.

AOP and Instrumentation: Spring's AOP module provides an AOP Alliance-compliant aspect-oriented programming implementation allowing you to define, for example, method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated.

The separate Aspects module provides integration with AspectJ.

The Instrumentation module provides class instrumentation support and class-loader implementations to be used in certain application servers.

Test: The Test module supports the testing of Spring components with JUnit or TestNG. It provides consistent loading of Spring ApplicationContexts and caching of those contexts. It also provides mock objects that you can use to test your code in isolation.

2.4 Related Work

3 Requirements

As described in section 1.2, the objective of this thesis is to design and develop a generic contextual content management platform, which supports the correlation of context information and multimedia content, content discovery and content distribution. In the following sections, the problem statement is discussed furthermore providing two instructive use cases. The use cases illustrates the functional assets of the framework and facilitate a deduction of its non-functional requirements.

3.1 Problem Statement

The evident growth of multimedia content offered by highly competitive providers alongside the technological progress concerning portable internet-ready devices(e.g. iPhone, iPad) calls for a more efficient correlation between content requests on the one hand and personalized search results and discovery mechanisms provided by the framework on the other. The challenge is to provide users with technical means for rapid and instant access to relevant, trustworthy multimedia content information and enriched personalization.

TODO

3.2 Scenarios

For better comprehension of the functionality of the proposed platform, two examples are given in this section as use case scenarios.

Mobile capturing of live event A user - as a content provider - captures a live event (e.g. demonstration, car race, marathon, Tour de France, etc.) using an application on a GPS capable smart phone. While filming, the application also collects context information (e.g. location, acceleration, temperature, time, etc.). Later the user uploads the captured content with its context information to the platform.

Let's consider a video of the Tour de France as an example for the uploaded content. Any consumer, who is interested in a specific uploaded video that has been taken in a specific place on the road of the tour (e.g. Les Essarts: town

which is located in western France), searches for the video by specifying some related information such as time range, location and "Tour de France" as search string. The platform will then give the user a list of all videos that match the specified criteria. The user can select any of the listed videos and begin streaming.

Restaurant guide Another approach would be a domain oriented search (e.g. restaurants, gas-station's, public libraries...). Considering a hungry user looking for a suitable restaurant; the restaurant guide will help users find facilities based on search criteria e.g name, place, cuisine, ratings ... etc. and then display images or videos of the selected restaurant and other information e.g price list, daily menu etc.

3.3 Functional requirements

In 'normal' applications, the functional requirements serve the purpose of describing what the system should do. This applies to both internal processes, and the interaction of the system with its environment. These requirements are derived commonly from use cases. Frameworks are different, identifying functional requirements is more difficult. Frameworks usually do not address specific use cases but are supposed to be open for varied scenarios. The functional requirements of the framework are therefore rather abstract.

R1 User Management Administrators of the framework can grant access to other administrators or providers, thereby ensuring a simple user management. Once a provider has been registered, he can administrate the flow and accessibility of multimedia content via his respective applications.

Applications Management The framework should provide an easy mechanism for creating and deleting applications. When a provider wishes to create a new application he sends a request including global configurations. A provider may also grant access to other providers to use his applications. This access either grants full administrative control over the applications or can be limited to uploading/downloading and searching activities.

Content & Context Data Store The framework should furthermore provide a mechanism for creating data stores for multimedia contents and their related context. In order to ensure legitimate context-based search results, the framework facilitates the correlation between content and its related context.

Content Discovery In order for the provider to discover context-based contents, an adjustable search engine is required. To this end, the provider defines the parameters relevant to potential search requests. The provider can respond to new challenges and refine search options as time goes on by adding new parameters to the original set.

Content Adaptation Based on the global configuration of the application, the framework should support content transcoding. Transcoding refers to optimizing processes, e.g. quality adjustments for efficient use in varying networks (mobile, Wifi), size adjustment for individual displays in the area of video contents or Word to PDF conversions for textual contents.

Content Distribution The framework should support content delivery to most widespread internet-able devices. It should also optimize delivery by streaming contents from the nearest available server, thereby minimizing the network latency and reducing bandwidth costs.

3.4 Non functional requirements

Non-functional requirements are mainly related to quality aspects of a system. As a implementation of the design presented in Chapter 4, which is considered in this work as a prototype, the non-functional requirements play a subordinate role. However, some non-functional requirements have quite an influence on fundamental architectural decisions. So it is nevertheless important to analyze these requirements. The following section outlines the non-functional requirements for the development of the framework.

3.4.1 Usability

Since this thesis concerns the development of a framework and not a concrete application or GUI this quality feature's applicability is limited. However there are certain ease-of-use requirements constituting the degree of effort needed to comprehended, evaluate and effectively use the software that are relevant.

All functionalities of the framework shall be accessible in a simple way. From a developer's point of view, who in a sense represent the "user" of the framework, it can be said that a good structure and readability of the source code is desirable. Changes and enhancements to the framework shall always be restricted to as few logical components as possible. More importantly, however, the standard use of the framework shall not require deep knowledge of the internal structure of the framework's components. Creating a new application for instance, shall be

possible without further knowledge of the framework via the external interface of the framework.

3.4.2 Efficiency

The efficiency describes the response time for inquiries, as well as the consumption of resources. The framework shall be capable to serve multiple applications simultaneously. The creation of an additional application shall affect the performance of the overall system only marginally.

3.4.3 Changeability

3.4.4 Scalability

4 Design

Against the background of the requirements described previously in chapter 3, this chapter provides a design concept for the framework. Section 4.1 will describe a basic architectural concept of the framework to be refined in section 4.2 and 4.3 which provide a more detailed description of the framework's components and their interactions. The final section reviews the parity between the requirements from chapter 3 and the corresponding architectural components.

4.1 Architecture Overview

Figure 4.1 shows an architecture overview of the framework, which consists mainly of six components. The components with three dots(...) in them mean that the framework can be later extended with new components for adding new services to the framework like SUBSCRIBE/NOTIFICATION etc.

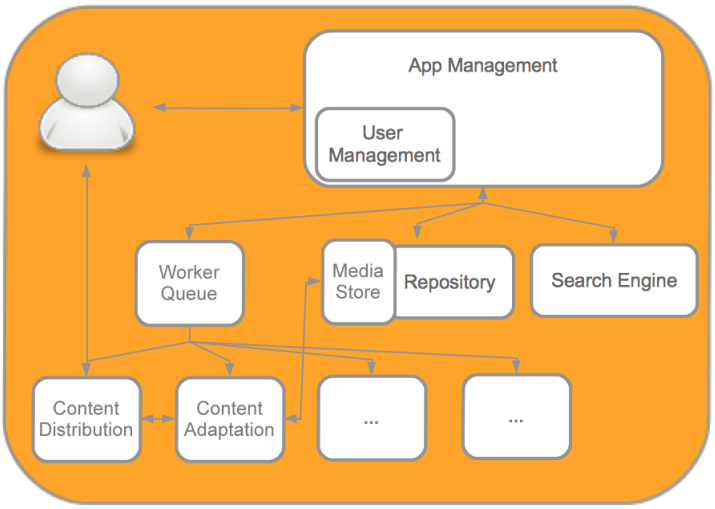

Figure 4.1: Architecture Overview

4.2 Framework Components

The App Management component is the main entry for each developer to interact with the framework. These developers needs to authenticate them self in order to use the framework and therefore a appropriate User Management is needed. Furthermore the Repository/Media Store is needed in order to store the data an the multimedia files. For efficient data discovery, the Search Engine component is required. The task of the Content Adaptation component is to convert the stored files to other formats and then upload these converted files to the Content Distribution component which then serve them to various devices.

Those components are described in more details in the next subsections.

4.2.1 App Management

The App Management component is the core of the entire framework and it interacts with almost all components in the framework. The following subsections describe the design decision that have been made for implementing this component.

JSON

The data format to interact with the framework is JSON. It has been chosen due its low-overhead comparing with XML and it is also the data format for many of the known NoSQL systems, which one of them will be used in the framework.

User Management

In order to allow only registered developers to use the framework, a User Management design concept is described in the following.

The User Management component provides two levels of management. The first one is to provide a role based user management for using the entire framework. The role can be either an admin or a normal user - which here means a developer who use the framework. Both admin and user are allowed to use the framework i.e. creating new applications, modeling the app and storing, getting or deleting contents etc. Only admins can add a new admin or user.

The second level of the User Management component provides a mechanism for managing who can modify an existing app or store/get contents from this app. The user who created the app is theoretically the owner of it and he is the only one who is allowed to add other users for using his app. Thereby he can add users with the same rights as his rights, meaning they can modify the whole app, deleting contents etc., or add other users, who only allowed to store/get/search contents. For example this is useful for a developer, who has deployed/configured

a new app and then lets a service provider to use his app by only storing, getting, searching the contents in this app.

REST Interfaces

For interacting with the outside world, a standard and well defined interface is needed. The decision for choosing REST as an API interface is due to its flexibility, simplicity, less bandwidth usage and very easy way to scale for large deployment, see section 2.3.1 for more detailed comparison.

Following paragraphs describes the various interfaces needed in order to be able to use the framework:

Framework User Management Interface /users : This interface provide the first level of user management described in 4.2.1. Only admins are allowed to use this interface for managing users.

- **Create:** The method *addUser* requires following parameters, *username*, *password*, *firstname*, *lastname*, *email* and *role*. The role parameter is an integer and specify if the user has ADMIN - 1 - or USER - 2 - role. These parameters is passed through an URI parameters, see following example for adding a new admin.

```
POST https://user1:pass@107.23.121.185:8080/cccd/
/users?username=frank&password=pass2&firstname=frank&
lastname=schulz&email=frank21@yahoo.de&role=1
```

- **Delete:** The method *deleteUser* requires only *username* as a parameter. It is not allowed to delete the *username* if he is the last admin in the framework, else no one can admin it.

```
DELETE https://user1:pass@107.23.121.185:8080/cccd/
/users?username=frank
```

App interface /app/ : This interface provides the four CRUD operation, which described bellow, to allow developers to interact with there apps.

- **Create(POST):** This method *createApp* creates a new app and it requires a name and global configuration parameters for the app. These configuration parameters can be for example a mandatory secret word to be later used for securing the contents which belongs to this app or a list of video formats, which will be then used to transcode each uploaded video within this app

in order to support various devices i.e. iPad, iPhone, PC and etc. Listing 3 shows a JSON example for creating a new app with **vod1** as a name.

```
POST https://user1:pass@107.23.121.185:8080/cccd/app/vod1
```

Payload:

```
{
"secret":"pass123",
"video_formats":"cell_4x3_150k,wifi_4x3_640k,wifi_4x3_1240k"
}
```

Response:

```
{
"ok": "1",
"debug": "app: vod1 created"
}
```

Listing 3: Creating a new app

- **Read(GET):** There are two methods in this App Interface, which can be consumed through a HTTP GET, first one *listApps* is for listing all apps which belongs to the user, see listing 4 as an example.

```
GET https://user1:pass@107.23.121.185:8080/cccd/app
```

Response:

```
{
"data":["app3","vod","vod1","app1"],
"ok":"1"
}
```

Listing 4: Listing all apps which belong to a user

The second method *checkAppStatus* returns data that reflects the amount of storage used and data contained in a specific app, as well as object, collection, and index counters. This data can be used to check and track the state and storage of a specific app, the example below will returns these data for an app called *vod1*.

```
GET https://user1:pass@107.23.121.185:8080/cccd/app/vod1
```

- **Update(PUT):** This method *updateApp* is for updating/adding the configuration parameters of the app. i.e changing the secret word or adding new configuration parameters which can be needed from other components, i.e. adding the user credentials for a Justin.TV account in order to stream a video to it.
- **Delete(DELETE):** This method *deleteApp* delete the app along with its related contents which might be in the Repository, Media Store or in the Content Distributor component.

App User Management Interface /app/appName/users : This interface provide the second level user management described in 4.2.1. The users table is saved on the repository within the app. In order to add a user to an app with WRITE or only READ rights, he must be already registered in the framework as a valid user.

- **Create:** The method *addUser* takes following parameters, *appName*, *username* which needs to be added to the app and which rights will it has *readonly*. The *username* and *readonly* parameters are passed through URI parameters, see following example.

```
POST https://user1:pass@107.23.121.185:8080/cccd/app/vod1
/users?username=frank&readonly=true
```

- **Delete:** The method *deleteUser* takes two parameters, namely *appName* and the name of the user *username* which needs to be deleted from the app. It is not allowed to delete the *username* if he is the last user who has WRITE rights, else no one else can admin the app.

```
DELETE https://user1:pass@107.23.121.185:8080/cccd/app/vod1
/users?username=frank
```

Collection Interface /app/appName/collections/ : The collection within an app can be compared with tables in SQL systems and it contains the real data, i.e. the metadata for a content. This interface provide only Create, Read, and Delete operations.

- **Create:** The method *createCollection* requires two argument, namely the app name and the name of the new collection. This method creates automatically an empty index in the search engine, which can be later used for searching within this collection. The name of the created index is based on

the name of the app combined with '_' and the name of the collection, i.e if the name of the app is *vod1* and the name of the collection is *collection1*, the created index name would be *vod1_collection1*.

- **Read:** The method *listCollections* is for listing all collections within an app, see listing 5 for an example.

```
GET https://user1:pass@107.23.121.185:8080/cccd/app/vod1
/collections
```

Response:

```
{
"data":["collection1","collection2"],
"ok":"1"
}
```

Listing 5: Listing all collections within an app

- **Delete:** Deleting a collection would delete the entire data in it and also delete its index in the Search Engine. Furthermore the data/files which are in the Content Distribution component, which belong to this collection will be deleted too.

Mapping Interface /app/appName/collections/collectionName/mapping : Before adding data to any collection, fields should be mapped to a proper object, i.e. string, integer, date, array or geo_point.

cite http://www.elasticsearch.org/guide/reference/mapping/ Mapping is the process of defining how a document should be mapped to the Search Engine, including its searchable characteristics such as which fields are searchable and if/how they are tokenized. By default, there is no need to define an explicit mapping, since one is automatically created and registered when a new type or new field is introduced and have sensible defaults. Only when the defaults need to be overridden must a mapping definition be provided. Only Get and Update are provided through this interface.

- **Read:** The method *listAllMappings* takes as parameters the *appName* and the *collectionName* and returns the entire mapping list for this collection.

```
GET https://user1:pass@107.23.121.185:8080/cccd/app/vod1
/collections/collection1/mapping
```

Payload:

```
{
"data": {
        "sourceAsMap": {
                "properties": {
                        "date": {
                                "type": "date",
                                "format": "dateOptionalTime"
                                },
                        "name": {
                                "type": "string"
                        },
                        "owner": {
                                "type": "string"
                        }
                }
        }
},
"ok": "1"
}
```

- **Update:** The method *updateMapping* takes as parameters the *appName*, the *collectionName* and the mapping description *body* as payload in JSON format and allows to register specific mapping definition for a specific collection.

```
PUT https://user1:pass@107.23.121.185:8080/cccd/app/vod1
/collections/collection1/mapping
```

Payload:

```
{
        "collection1" : {
                "properties" : {
                        "name" : {"type" : "string"}
                }
        }
}
```

Document Interface /app/appName/collections/collectionName/doc : The repository of the framework is a document-based database system, and as a result,

all records, or data, in it are documents. Documents are the default representation of most user accessible data structures in the repository.

- **Create:** The method *createDocument* takes as parameters the *appName*, *collectionName* and the document it self as payload in JSON format *body* and saves this document in the repository and also in the search engine in order to make the document searchable.

```
POST http://abdul:abdul@107.23.121.185:8080/cccd/app/vod1
/collections/collection1/doc
```

Payload:

```
{
"name":"video in Berlin",
"date":"2013-01-01T01:12:12",
"owner":"Tom"
}
```

- **Read:** The method *getDocument* requires the *appName*, *collectionName* and an *objectId* and returns the correspondent document. The example below shows how to get the document with the *objectId* 512d3258e4b0acc3647858f2.

```
GET http://abdul:abdul@107.23.121.185:8080/cccd/app/vod1
/collections/collection1/doc/512d3258e4b0acc3647858f2
```

Response:

```
{
"data":{
        "_id":{
                        "machine":-458183485,
                        "timeSecond":1361916504,
                        "inc":1685608690,
                        "time":1361916504000,
                        "new":false
        },
        "name":"video in Berlin",
        "owner":"Tom",
        "date":"2013-01-01T01:12:12"
},
"ok":"1"
}
```

- **Update:** The method *updateDocument* requires the *appName*, *collectionName*, *objectID* and the JSON formatted payload *body*. The body contains the fields, which needs to be updated within a document.

```
POST http://abdul:abdul@107.23.121.185:8080/cccd/app/vod1
/collections/collection1/doc/512d3258e4b0acc3647858f2
```

Payload:

```
{
"name":"video in Potsdam"
}
```

- **Delete:** The method *deleteDocument* deletes the specific document from the repository and from the search engine too.

```
DELETE http://abdul:abdul@107.23.121.185:8080/cccd/app/vod1
/collections/collection1/doc/512d3258e4b0acc3647858f2
```

Buckets Interface /app/appName/buckets : Just like a bucket holds water, buckets - known in Amazon Web Services - are a container for files. The name of the bucket must be unique within an app. The four CRUD operations are described below.

- **Create:** The method *createBucket* requires two parameters, namely the app name and the bucket name and it creates a bucket within an app. The following example is for creating a bucket name *bucket1* within the app *vod1*.

```
POST https://user1:pass@107.23.121.185:8080/cccd/app/vod1
/buckets/bucket1
```

- **Read:** Within buckets interface there are two methods, which can be consumed through a HTTP GET. The first method *listBuckets* lists all buckets which belong to a specific app.

```
GET https://user1:pass@107.23.121.185:8080/cccd/app/bookstore
/buckets
```

The second method *listAllFilesInBucket* takes as parameters the *appName* and the *bucketName* and lists all the files which are in bucket *bucketName*.

```
GET https://user1:pass@107.23.121.185:8080/cccd/app/bookstore
/buckets/books
```

- **Delete:** The method *deleteBucket* deletes a bucket along with its files from an app

```
DELETE https://user1:pass@107.23.121.185:8080/cccd/app/bookstore
/buckets/books
```

Files Interface /app/appName/buckets/bucketName/filename : This interface manages files within a buckets.

- Create: The method *addFileToBucket* allows to upload a file to a bucket and it requires the parameters *appName*, *bucketName* and *filename* along with the file its self.
- Read: The method *getFileFromBucket* allows to get a file from a bucket and it requires also the parameters *appName*, *bucketName* and *filename*. An example for downloading a file named *test.pdf* from the bucket *books* and app *bookstore* is listed below.

```
GET https://user1:pass@107.23.121.185:8080/cccd/app/bookstore
/buckets/books/test.pdf
```

- Update: The method *updateFileInBucket* allows to update a file within a bucket and it requires also the parameters *appName*, *bucketName*, *filename* and the file its self. The update here is theoretically removing the old file and then adding the new one.

```
PUT https://user1:pass@107.23.121.185:8080/cccd/app/bookstore
/buckets/books/test.pdf
```

- Remove: The method *deleteFileFromBucket* deletes a file from a bucket.

Informative Response:

In order to know if a request to the framework succeeded or failed, the API should always send a tag in each response which shows either the request processed successfully or not, i.e. "ok":1 in case of success or "ok":0 in case of failure. Furthermore a configuration parameter will be implemented to enable more debug information, i.e. while creating a new app, if its name exist already, then the response include the error as a string i.e. "ok":0,"debug":"app name exist already".

4.2.2 Repository/Media Store

The repository/media store is accessible through the REST interfaces described above. The design decisions for this component are listed below.

- JSON data format: As described in section 4.2.1, the REST API data format is JSON and therefor the repository should be based on JSON format too in order to avoid unnecessarily data conversations.

- Schema-less: The framework is generic and should support wide variety of applications and therefore the repository should be based on schema less solutions in order to allow the application developer to store schema-less data.

- Replication: Data replication ensures redundancy, backup, and automatic failover.

- Distribution: While replication provide basic protection against single-instance failure, when all of the members of a replication group are in a single facility, the replication is still susceptible to some classes of errors in that facility including power outages, networking distortions, and natural disasters. To protect against these classes of failures, the repository/media store component should be easy to distribute in a geographically distinct facility or data center.

- Media Store: In order to store or retrieve contents, a file server is needed. This file server can be either as a component within the repository itself or as a separated component. In order to ensure the correlation between the content stored in the media store and its metadata within the repository, the URI of the content in the media store is embedded in its metadata in the repository.

4.2.3 Search Engine

The search engine is accessible through following REST interfaces, *search* and *mapping*. The first one is for searching withing a collection. The later is for configuring the mapping of fields within a collection. The design decisions for this component are mostly like the repository/media store component described in 5.2.2 like JSON data format, schema-less, easy to replicate and distribute.

4.2.4 Worker Queue

The main idea behind Work Queue or Task Queue component is to avoid doing a resource-intensive task immediately and having to wait for it to complete. Instead the task will be scheduled to be done later. The task get encapsulated as a message and get sent to a queue. A worker process running in other components like 4.2.5 or **??** will pop the tasks and eventually execute the job. When there is many of the same worker the tasks will be shared between them.

4.2.5 Content Adaptation

The Content Adaptation component is a worker process and it listens for new tasks from the Worker Queue. This component is responsible for transcoding or scaling videos. Furthermore it splits the transcoded/scaled videos into segments compatible to the Apple HLS.

Figure 4.2 illustrates how this component works internally. As soon as it receives a message notification from the Worker Queue component, which tells that a new video has been stored and it needs further processing i.e. transcoding or scaling, it fetches the video file from the media store and and then provides this file as an input for the media encoder. The media encoder transcodes and scales this video and returns a MPEG-2 transport stream which is used by the stream segmenter as an input, The stream segmenter brokes this stream into segments and saves these segments as a series of one or more .ts media files. The segmenter also creates an index file which contains a list of media files and metadata. This index file is saved as an .M3U8 playlist. The index file along with the .ts media files are uploaded then to the proper path in the Content Distribution component.

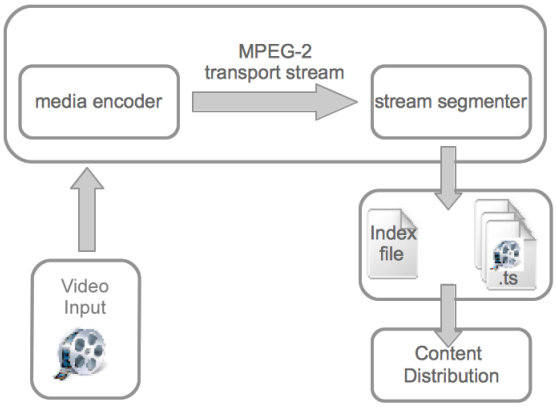

Figure 4.2: Content Adaptation Overview

4.2.6 Content Distribution

The Content Distribution component consists internally of two components. The first one is an HTTP server to serve contents to consumers, see figure 4.3. The second one is a worker process, which listens for messages notifications from the Worker Queue component. These messages tells when a new app is created or deleted. In case of a new created app, the worker process creates a new dictionary within the HTTP server and secures it with a secret word to grants access only to this app. And in case of a deleted app, the worker process deletes the dictionary for this app along with its entire contents.

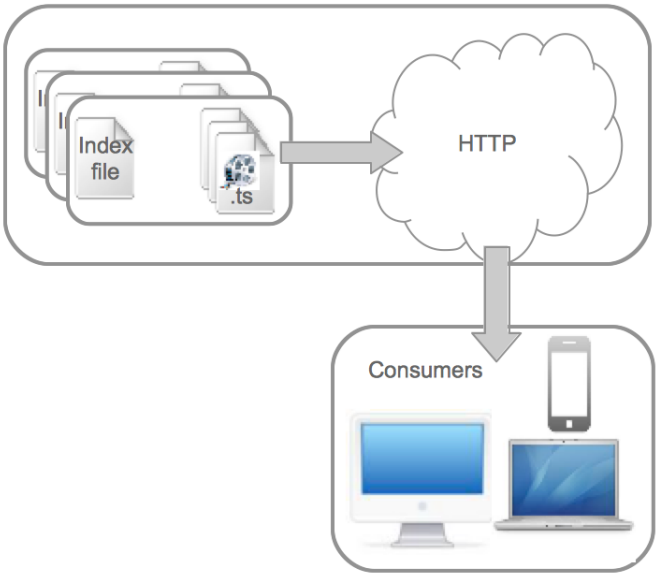

Figure 4.3: Content Distribution

5 Implementation

This chapter deals with the implementation of the design presented in Chapter 4. It was and is not the purpose of this work to implement a market-ready product. The focus is clearly on the conceptual part. To demonstrate the general realization of the proposed design, the implementation is done therefore only in the form of a prototype.

5.1 Tools & Technologies

This section gives an overview on the tools and technologies that have been used to simplify the implementation. The programming languages, which have been used to implement the framework, are Java and Python.

5.1.1 Tools

The following tools have been used to simplify, organize and test the implementation.

Eclipse Juno 4.2 IDE: Eclipse is one of the widely used IDE for Java. It makes it easier to develop Java applications.

Maven 3: Maven is an open source build automation tool developed by the Apache Software Foundation. It uses an XML file called *pom.xml* to describe the software project being built, its dependencies, the build order, directories, and required plug-ins. It comes with predefined targets for performing certain well-defined tasks such as compilation of code, its packaging and how and where to deploy the project.

Advanced REST client: Advanced REST client is a plugin within the Chrome browser and can help developers to create and test custom HTTP requests. It has been used mainly to test the different REST API, which have been developed within the framework.

5.1.2 Technologies

In order to not reinvent the wheel again, most of the components in the framework are based on existing open source technologies.

Spring Framework 3.1.1: As described in 2.3.5, the Spring framework is a lightweight solution and a very good base for building enterprise-ready applications. It provides an incredibly powerful and flexible collection of technologies and projects to improve enterprise Java application development. The following are some projects from the Spring framework, which have been used in developing the framework.

Spring Security 3.1.1: Spring Security is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications.

Spring Data MongoDB 1.1.0: Spring Data for MongoDB is part of the umbrella Spring Data project which aims to provide a familiar and consistent Spring-based programming model for for new datastores while retaining store-specific features and capabilities. The Spring Data MongoDB project provides integration with the MongoDB document database.

Spring AMQP 1.1.3: The Spring AMQP project applies core Spring concepts to the development of AMQP-based messaging solutions. It provide a "template" as a high-level abstraction for sending and receiving messages.

MongoDB 2.2.3: As described in 2.3.2, MongoDB is a free and open-source document-oriented database and is completely schema-free and manages JSON-style documents.

Elasticsearch 0.20.4: Elasticsearch is an open-source, distributed, RESTful, search engine built on top of Apache Lucene.Its data model roots lie with schema-free and document-oriented databases, and as shown by the NoSQL movement, this model proves very effective for building applications.

RabbitMQ 3.0.1: RabbitMQ is an open-source message broker, which implements the AMQP standard. It provides robust messaging services for applications and is reliable and highly scalable.

NginX 1.2.6: Nginx- engine-x pronounced - is a free, open-source, high-performance HTTP server. Unlike traditional servers, Nginx does not rely on threads to handle requests. Instead it uses a much more scalable event-driven (asynchronous) architecture. This architecture uses small, but more importantly, predictable amounts of memory under load.

FFmpeg 0.9.2: FFmpeg is the leading multimedia framework, able to decode, encode, transcode, mux, demux, stream, filter and play pretty much anything that humans and machines have created. It is an open source project licensed under LGPL version 2.1.

5.2 Framework Components

This section describes how each component in the framework got implemented and realized.

5.2.1 App Managment

The *app management* component is developed completely in Java and it is based on the Spring framework. Figure bla shows the project structure of this component withing the Eclipse IDE.

As shown in figure bla, the project consist of three java packages, namely *de.fhg.fokus.ngni.cccd.model*, *de.fhg.fokus.ngni.cccd.rest* and *de.fhg.fokus.ngni.cccd.services*, and also four configurations file, which are, the maven configuration file *pom.xml*, the *web.xml* file, the logging configuration file *log4j.xml* and the Spring configuration file *cccd-config.xml*.

The maven configuration file *pom.xml* contains information about the project and configuration details used by Maven to build the project and also manages the dependencies which are needed for building the project.

Within the *web.xml* file, one can set how to map a specific URL to a particular servlet and Spring provides this in the following form:

```
<web-app>
  <servlet>
    <servlet-name>cccd</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <init-param>
      <param-name>
        contextConfigLocation
      </param-name>
      <param-value></param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>cccd</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

In the above example at the servlet-mapping tag, this says that all URL("/") being requests should be handled by the *cccd* servlet. And what is the *cccd* servlet and where to find it is declared in the servlet tag which tells Tomcat which Java class it should resolve this *cccd* servlet to. Normally in none Spring applications one could just directly specify a class which inherits from the HttpServlet class. In Spring, however, this is where it actually enter the Spring Framework. Instead of defining the class, which needs to be executed for this servlet directly, one need to specify only the *org.springframework.web.servlet.DispatcherServlet*. From this point onwards, the request and response are known by the Spring framework, so that one can apply Spring pre-processing and post-processing with special Spring modules, i.e. Security, Aspect Oriented Programming and so on.

5.2.2 Repository

MongoDB

5.2.3 Search Engine

ElasticSearch

5.2.4 Content Adaptation

FFmpeg

5.2.5 Content Distribution

HTTP-Live-Video-Stream-Segmenter-and-Distributor

NginX

5.2.6 Application Messaging

RabbitMQ

5.2.7 User API

SPRING DATA - REST

5.3 Components Integration and Configuration

5.4 REST API

6 Evaluation

6.1 Test Environment

6.2 Test Scenarios

6.2.1 Usability

6.2.2 Performance

7 Conclusion

Bibliography

[1] Web services architecture. URL `http://www.w3.org/TR/ws-arch/`.

[2] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, jun 2007. ISSN 1743-8225. doi: 10.1504/IJAHUC.2007.014070.

[3] P.J. Brown, J.D. Bovey, and Xian Chen. Context-aware applications: from the laboratory to the marketplace. *Personal Communications, IEEE*, 4(5):58 –64, oct 1997. ISSN 1070-9916. doi: 10.1109/98.626984.

[4] Cisco. Cisco visual networking index: Forecast and methodology, 2010-2015. http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf.

[5] Anind K. Dey. Context-aware computing: The cyberdesk project. In *AAAI 1998 Spring Symposium on Intelligent Environments*, pages 51–54, Palo Alto, 1998. AAAI Press. URL `http://www.cc.gatech.edu/fce/cyberdesk/pubs/AAAI98/AAAI98.html`.

[6] Anind K. Dey and Gregory D. Abowd. Towards a better understanding of context and context-awareness. In *Workshop on The What, Who, Where, When, and How of Context-Awareness (CHI 2000)*, The Hague, The Netherlands, April 2000. URL `http://www.cc.gatech.edu/fce/contexttoolkit/`.

[7] Jie Ding and Ning Li. A distributed adaptation management framework in content delivery networks. In *Wireless Communications, Networking and Mobile Computing (WiCOM), 2011 7th International Conference on*, pages 1 –4, sept. 2011. doi: 10.1109/wicom.2011.6040622.

[8] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[9] EC FIArch Group. Fundamental limitations of current internet and the path to future internet. http://ec.europa.eu/information_society/activities/foi/docs/current_internet_limitations_v9.pdf.

[10] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993. URL `http://tomgruber.org/writing/ontolingua-kaj-1993.pdf`.

[11] John Mccarthy. Notes on formalizing context. In *IJCAI*, pages 555–562, San Mateo, California, 1993. Morgan Kaumann. URL `http://dblp.uni-trier.de/rec/bibtex/conf/ijcai/McCarthy93`.

[12] Oriana Riva. A conceptual model for structuring context-aware applications. In *Fourth Berkeley-Helsinki Ph.D. Student Workshop on Telecommunication Software Architectures*, 2004.

[13] N Ryan. Mobile computing in a fieldwork environment: Metadata elements. Project working document, version 0.2, 19997.

[14] N. Ryan, J. Pascoe, and D. Morse. Enhanced reality fieldwork: the context-aware archaeological assistant. In V. Gaffney, M. van Leusen, and S. Exxon, editors, *Computer Applications and Quantitative Methods in Archaeology (CAA 97), Oxford*, 1997. URL `http://www.cs.ukc.ac.uk/research/infosys/mobicomp/Fieldwork/Papers/CAA97/ERFldwk.html`.

[15] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. Proceedings., Workshop on*, pages 85 –90, dec 1994. doi: 10.1109/MCSA.1994.512740.

[16] B.N. Schilit and M.M. Theimer. Disseminating active map information to mobile hosts. *Network, IEEE*, 8(5):22 –32, sept.-oct. 1994. ISSN 0890-8044. doi: 10.1109/65.313011.

[17] Quan Z. Sheng and Boualem Benatallah. Contextuml: A uml-based modeling language for model-driven development of context-aware web services development. In *Proceedings of the International Conference on Mobile Business*, pages 206–212, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2367-6. doi: 10.1109/ICMB.2005.33. URL `http://portal.acm.org/citation.cfm?id=1084013.1084215`.

[18] Thomas Strang and Claudia L. Popien. A context modeling survey. In *UbiComp 1st International Workshop on Advanced Context Modelling, Reasoning and Management*, pages 31–41, Nottingham, September 2004. URL `http://citeseer.ist.psu.edu/712164.html`.

[19] Roy Want, Andy Hopper, Veronica Falcao, and Jonathan Gibbons. The active badge location system. *ACM Trans. Inf. Syst.*, 10(1):91–102, 1992. URL `http://dblp.uni-trier.de/db/journals/tois/tois10.html#WantHFG92`.

[20] Wikipedia. Object-oriented modeling, 2012. URL `http://en.wikipedia.org/wiki/Object-oriented_modeling`. [Online; accessed 01.02.2013].

[21] Youtube. press statistics. http://www.youtube.com/t/press_statistics.

[22] Pinar Öztürk and Agnar Aamodt. Towards a model of context for case-based diagnostic problem solving. In *IN CONTEXT-97; PROCEEDINGS OF THE*, pages 198–208, 1997.