Fakultät Informatik
Hochschule Reutlingen
Alteburgstraße 150
D-72762 Reutlingen

Masterthesis

Bestimmung der Dokumentenähnlichkeit basierend auf Bayessche Statistik für eine Big-Data Information Retrieval Lösung

Elisabeth Agnes Mpessa Enangue

Studiengang: Services Computing

Betreuer Hochschule: Prof Dr.-Ing Christian Decker
Betreuer Unternehmen: Steve Strauch
Abgabetermin: 31. Juli 2018

Abstract

In the last years Cloud computing has become popular among IT organizations aiming to reduce its operational costs. Applications can be designed to be run on the Cloud, and utilize its technologies, or can be partially or totally migrated to the Cloud. The application's architecture contains three layers: presentation, business logic, and data layer. The presentation layer provides a user friendly interface, and acts as intermediary between the user and the application logic. The business logic separates the business logic from the underlaying layers of the application. The Data Layer (DL) abstracts the underlaying database storage system from the business layer. It is responsible for storing the application's data. The DL is divided into two sublayers: Data Access Layer (DAL), and Database Layer (DBL). The former provides the abstraction to the business layer of the database operations, while the latter is responsible for the data persistency, and manipulation.

When migrating an application to the Cloud, it can be fully or partially migrated. Each application layer can be hosted using different Cloud deployment models. Possible Cloud deployment models are: Private Cloud, Public Cloud, Community Cloud, and Hybrid Cloud. In this diploma thesis we focus on the database layer, which is one of the most expensive layers to build and maintain in an IT infrastructure. Application data is typically moved to the Cloud because of , e. g. Cloud bursting, data analysis, or backup and archiving. Currently, there is little support and guidance how to enable appropriate data access to the Cloud.

In this diploma thesis the we extend an Open Source Enterprise Service Bus to provide support for enabling transparent data access in the Cloud. After a research in the different protocols used by the Cloud providers to manage and store data, we design and implement the needed components in the Enterprise Service Bus to provide the user transparent access to his data previously migrated to the Cloud.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

List of Listings

1. Einleitung

1.1. Motivation

A multi-tenant aware architecture in a Cloud environment is one of the main keys for profiting in a Cloud infrastructure. Virtualization and simultaneously usage of resources by multiple users allow Cloud providers to maximize their resources utilization. However, a multi-tenant environment requires isolation between the different users at different levels: computation, storage, and communication [**?**]. Furthermore, the communication to and from the Cloud infrastructure must support different protocols.

Das Problem habe ich im Abschnitt **??** beschrieben.

Migration of an application to the Cloud can be divided into four different migration types: component replacement with Cloud offerings, partial migration of functionalities, migration of the whole software stack of the application, and cloudifying the application [**?**]. In this diploma thesis we focus on the needed support when the first migration type takes place. For example, due to an explosive data growth a tenant may decide at some point in time to migrate and host his local business data in a Cloud storage infrastructure, while maintaining his application's business logic on-premise. Bachmann provides a prototype which assists the tenant during the data migration process from a local storage system to a Cloud data store, and between Cloud data stores [**?**]. However, as described before his work covers the migration process, but it does not provide data access or data modification after the migration.

An Enterprise Service Bus is a central piece in a Platform-as-a-Service (PaaS) environment for providing flexible and loosely coupled integration of services as well as multi-tenant aware and multi-protocol communication between services. In this diploma thesis we extend the multi-tenant aware prototype of an Enterprise Service Bus (ESB) produced in [**?**], [**?**], and [**?**]. The final prototype must provide multi-tenant and multi-protocol communication support, and transparent Cloud data access to tenants who migrate their application data partially or completely to the Cloud.

The use of an intermediate component in data transfer may have a negative impact on the overall data processing in an application. For this reason, we provide an evaluation using example data from an existing TPC benchmark in order to investigate the impact on the performance and to propose future optimizations [**?**].

1.2. Problemstellung

IT industries aim to reduce their budget in expensive hardware investment and maintenance, e.g. Database Management Systems, while maintaining data access and persistency over time. In order to fulfill a budget reduction while maintaining their data and database system requirements, the data can be migrated to different Cloud storage providers available nowadays in the market. Considering the application's migrations types discussed in [**?**], migrating local data to the Cloud, and then accessing it from the application's hosted on-premise, can be considered as a *partial or complete replacement of components with Cloud offerings*. Such migration requires a reconfiguration, rewiring, and adaptation activities on both the migrated and non-migrated components.

The *Cloud Data Migration Application* assists the user in the migrating decision and process of local data to a Cloud storage provider, or from two different Cloud storage providers [**?**]. It contains a registry of the different available Cloud data stores, e.g. Amazon Dynamo DB [**?**], Amazon RDS [**?**], Google Cloud SQL [**?**], and detects possible incompatibilities between the source and target data sources prior to the migration process.

Migration of data can be either seen as the migration of only the Data Layer, or as a part of the migration of the whole application [**?**]. The approach we consider as the start point in this diploma thesis is the migration of the Data Layer, which contains two sublayers: the *Database Layer (DBL)* and the *Data Access Layer (DAL)*. The DBL contains the database information, e.g. location, access credentials, etc., and gives the DAL a direct communication support to the database server. The DAL provides simplified access support to the upper layers of the data stored in a backend database. However, migrating the Data Layer to the Cloud implies adaptations and rewiring of the original application. One of our main goals in this diploma thesis is to minimize the needed adaptations in the non-migrated and migrated components by providing a transparent access to the user's data migrated to a Cloud datastore.

As the Enterprise Service Bus is an established integration middleware for services in Service-Oriented Architectures (SOA), and due to its multi-protocol support and reliable internal messaging routing, we use it as a central piece for providing a multi-tenant aware, transparent and reliable communication support between the on-premise and the off-premise layers of the application.

As shown in Figure 1.1, the Cloud-Enabled Data Access bus provides access support between the hosted on-premise, and off-premise application's layers. Its main goal is to provide communication isolation between different applications and users, and maintain the transparency that the DL provided before the migration to the upper layers of the application's architecture. Support must be provided for two different databases types: MySQL and NoSQL databases, and between different providers. A tenant who migrates its data, e.g. to the Google SQL Datastore in Google App Engine, as shown in Figure 1.1, must be able to access his data with minimum adaptations of the components. Furthermore, storing or retrieving data whose storage is divided into multiple datasources requires a dynamic routing between backend data stores. Compatibility between different SQL and NoSQL databases must be also ensured.

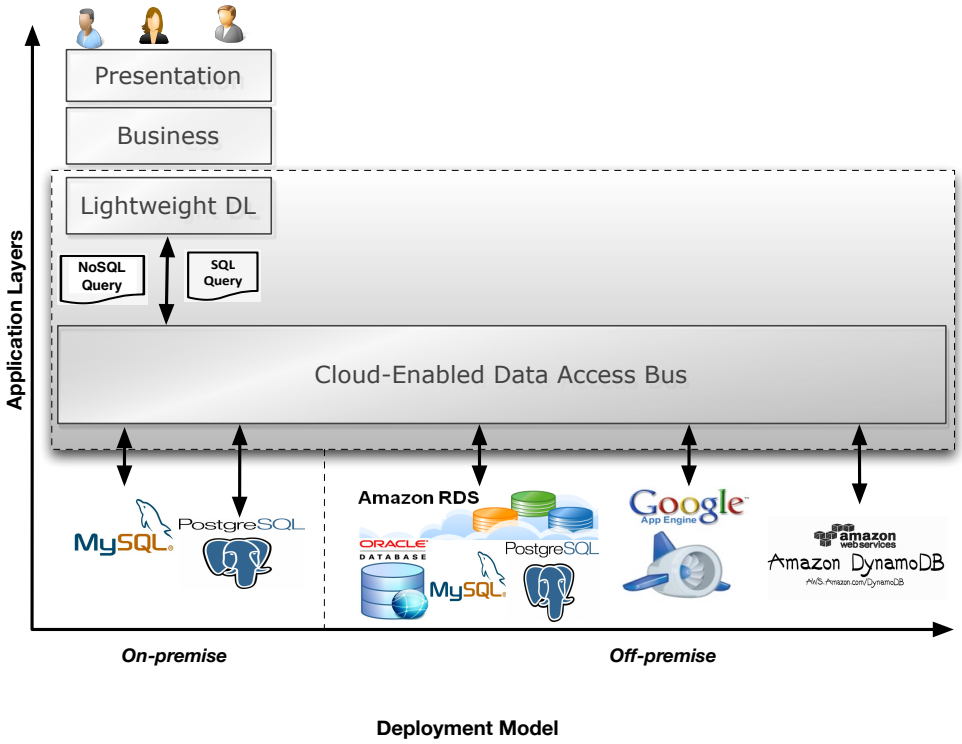

Figure 1.1.: Migration Scenario to be filled

However, query and data transformation between different data sources types is out of the scope of this diploma thesis.

1.3. Zielsetzung

In the following section we list the definitions and the abbreviations used in this diploma thesis for understanding the description of the work.

Definitions

List of Abbreviations

The following list contains abbreviations which are used in this document.

API	Application Programming Interface
BC	Binding Component
DBaaS	Database-as-a-Service
DBMS	Database Management System
ESB	Enterprise Service Bus
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure-as-a-Service
JBI	Java Business Integration
JMS	Java Message Service
JMX	Java Management Extensions
JNDI	Java Naming and Directory Interface
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MDBS	Multidatabase System
MEP	Message Exchange Patterns
NIST	National Institute of Standards and Technology
NM	Normalized Message
NMF	Normalized Message Format
NMR	Normalized Message Router
NoSQL	Not only Structured Query Language
OSGi	Open Services Gateway initiative *(deprecated)*
PaaS	Platform-as-a-Service

SA	Service Assembly
SaaS	Software-as-a-Service
SE	Service Engine
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol *(deprecated)*
SQL	Structured Query Language
STaaS	Storage-as-a-Service
SU	Service Unit
TCP	Transmission Control Protocol
WSDL	Web Services Description Language
XML	eXtensible Markup Language

1.4. Aufbau der Arbeit

The remainder of this document is structured as follows:

- **Fundamentals, Chapter ??:** provides the necessary background on the different concepts, technologies, and prototypes used in this diploma thesis.
- **Related Works, Chapter ??:** discusses relevant State of the Art and positions our work towards it.
- **Concept and Specification, Chapter ??:** functional and non-functional requirements are discussed in this section.
- **Design, Chapter ??:** gives a detailed overview of the differerent component's architecture, and the needed extensions to the already existing ones.
- **Implementation, Chapter ??:** the implemented components, as well as the necessary extensions or changes are detailed in this section from the point of view of coding and configuration.
- **Validation and Evaluation, Chapter ??:** in this chapter we test the final prototype based on the scenario described in this document.
- **Outcome and Future Work, Chapter ??:** we provide a conclusion of the developed work and investigate some ideas in which this diploma thesis can be extended.

2. Grundlagen, Analyse von Use Cases zur Dokumentähnlichkeitsbestimmung

In this chapter we give an explanation about the technologies and concepts this diploma thesis relies on. We start describing the fundamental concepts and introduce the components and prototypes that form the basis of our work.

2.1. Informationsrückgewinnung

In the last decades our world has become more and more interconnected. This interconnection added to the increase of the available bandwidth and the change in business models have forced IT Systems to fulfill its demands, leading to its reorganization into a public utility which offers public services, like water, electricity, etc. The National Institute of Standards and Technology (NIST) defines Cloud computing as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [**?**]. The Cloud computing model is composed of five characteristics:

1. On-demand self-service: a Cloud user consumes the Cloud provider's computing capabilities automatically without the need of human interaction.
2. Broad network access: computing capabilities are available via the network and can be accessed using standard mechanisms.
3. Resource pooling: computing capabilities in the Cloud provider side are virtualized to serve multiple consumers simultaneously using a multi-tenant model. The Cloud consumer generally has no sense of the provided resources.
4. Rapid Elasticity: computing and storage resources can be dynamically (and in some cases automatically) provisioned and released to respond to the actual consumers' demand.
5. Measured Service: resources' usage is monitored and measured in a transparent way to the Cloud consumer and provider for control and optimization purposes.

The control that the Cloud consumer has over the computer resources in a Cloud provider infrastructure is defined in three service models: *Software-as-a-Service (SaaS)*, *Platform-as-a-Service (PaaS)* and *Infrastructure-as-a-Service (IaaS)*. *SaaS* provides to the Cloud consumer access and usage of Cloud provider's applications running on a Cloud infrastructure. The consumer has no control over the underlying infrastructure where the application he uses is deployed. The customer can only control individual application's configurations during his

usage of it. *PaaS* provides the customer with the needed capabilities to deploy applications which's programming language, required libraries, services and tools are supported by the provider. The consumer has no control over the underlying infrastructure where he deploys the application. *IaaS* is the model which gives most control to the consumer. Thus, the consumer is able to deploy and run arbitrary software and has the control over operating systems, storage and deployed applications, but has no management or control on the underlaying Cloud infrastructure.

Although the three service models described above provide both data computation and storage capabilities for the consumer, they do not provide to the customer the possibility to directly and uniquely purchase access of storage services. In this diploma thesis we concentrate in two concrete models: *Database-as-a-Service (DBaaS)* and *Storage-as-a-Service (STaaS)*. Cloud storage providers target a selected number of consumers, who process their data on-premise, but do not want to cover the expenses of a local database system, or a backup system, among others. The Cloud storage model alleviates the need in organizations to invest in database hardware and software, to deal with software upgrades, and to maintain a professional team for its support and maintenance [**?**]. DBaaS and STaaS can be considered quite similar, except for one of their main distinction characteristics: their access interface. The former is the most robust data solution offered as a service, as it offers a full-blown database functionality. It can be accessed via the most common database protocols, such us MySQL, Oracle, etc, or by REST interfaces supporting Structured Query Language (SQL). Examples of this model are Amazon RDS [**?**] and Oracle Cloud [**?**]. On the other hand, the latter provides REST, SOAP over Hypertext Transfer Protocol (HTTP), or Web-based interfaces in order to perform the operations over the stored data [**?**]. Examples of this model are Amazon Dynamo [**?**] , Google App Engine Datastore [**?**], and Dropbox [**?**].

NIST defines four deployment models in Cloud computing. A private Cloud consists in a Cloud infrastructure which is provisioned exclusively for one organization and used by the members conforming the organization. It is comparable to processing facilities that are enhanced with the Cloud computing characteristics. A community Cloud is a Cloud infrastructure where its use is limited to organizations which share the same requirements. A public Cloud infrastructure can be accessed and used by the public. It is usually offered by Cloud service providers that sell Cloud services made for general public or enterprises. Some of the Cloud consumers may process and store information which requires more control over the infrastructure in which is located, or consume public Cloud computing resources during peak loads in their private Cloud infrastructure. The hybrid Cloud model combines two or more deployment models described above and the combination remains as a unique entity.

Cloud computing and Service-Oriented Architecture (SOA) are related styles at an architectural, solution and service level, according to IBM [**?**]. Cloud providers expose their Cloud infrastructure as services as part of a SOA solutions and the communication between Clouds in the Hybrid Cloud model described above can be compared to a SOA communication solution between enterprises. Cloud services are services that can be accessed by the Cloud consumers through the network. Therefore, we can deduce that the SOA model can be applied in the Cloud computing approach. As the ESB is the central piece of SOA, the need

of the ESB in a Cloud computing infrastructure as an integration middleware for the Cloud services is essential.

2.2. Bayessche Statistik

Weerawarana et al. define SOA as an specific architectural style that is concerned with loose coupling and dynamic binding between services [**?**].

In the last years communication between external components whose functionalities are exposed as services has been a hard task when there was not previous agreement on message protocols, data types and encoding, and used middleware technology. Due to the economic and technological growth needed, enterprises had to adapt the SOA paradigm in their existing IT Infrastructure. SOA provides the needed flexibility by building an architectural style with the following benefits: loose coupling, interoperability, efficiency, and standardization. The W3C group defines SOA as a form of distributed system architecture that is typically characterized by the following properties [**?**]:

- Logical view: the service's functionality is exposed, but not its internal logic.
- Message orientation: the internal structure of an agent is abstracted.
- Description orientation: a service is described by machine-processable meta data.
- Granularity: services tend to use a small number of operations with relatively large and complex messages.
- Network orientation: Services tend to be oriented toward use over a network.
- Platform neutral: Messages are sent in a platform-neutral, standardized format delivered through the interfaces.

SOA defines three main roles: requester, provider and broker and the four main operations: publish, find, bind, and invoke. The service provider provides access to services, creates a description of a service and publishes it to the service broker. The service requestor discovers a service by searching through the service descriptions located in the service broker. When the service which best fits to his needs is found, the discovering facility provides the concrete service endpoint and the consumer is responsible for binding to it. With this information, the requestor can then bind to the concrete service and finally execute a business activity [**?**]. The service broker provides support for service registration and binding.

The main component in a SOA is the ESB. The functionalities provided by a service bus can simplify the process (publication, discovery, binding, and invocation) and make it more transparent to provide an ease-to-use experience for a Web service based implementation of SOA [**?**]. Chappel defines its function as an intermediate connection provisioning of service providers with service consumers and thereby ensure decoupling of theses [**?**].

2.2.1. Definition und Hintergrund

2.2.2. Anwendungsgebiete und Nutzen

2.2.3. Modelle, Parameter und Überzeugungen

2.2.4. Die Wahrscheinlichkeit

2.2.5. Der Satz von Bayes

The flow of data and information is a key for driving business decisions in IT organizations [**?**]. Furthermore, the interaction between loosely coupled components within an organization or with third party organizations requires distributed systems mechanisms which provide communication support for different protocols, and reliability. SOA has fulfilled this main requirement by providing an integration environment with minimal (or any) integration efforts.

The ESB is the central component in SOA. It provides a loosely coupled, event-driven SOA with a highly distributed universe of named routing destinations across a multi-protocol message bus [**?**]. An ESB provides an abstract decoupling between connected applications by creating logical endpoints which are exposed as services and conform a multi-protocol environment, where routing and data transformation are transparent to the service connected to it. Furthermore, when using an ESB, in the first place, services are configured rather than coded, demanding minimal adaptation, implementation and maintenance efforts. The programmer just has to implement the binding to the logical endpoint exposed as a service. In the second place, ESB routing is based on a reliable messaging router. Applications don't need to include message system-failure forwarding mechanisms, to know which data formats are needed in the consumed services, or to care about future changes in applications or services the applications interact with. An ESB hides the complexity of orchestration between services in business processes.

Chappel defines the combination of loosely coupled interfaces and asynchronous interactions as a key concept of the bus terminology [**?**]. A user of the bus can access every service registered in the bus. For this purpose, it implements the SOA operations in order to make them transparent to the user who can therefore focus on: plugging to the bus and posting and receiving data from the bus. Furthermore, the ESB can form the core of a pervasive grid [**?**]. Services supported by an organization can be organized between the ESBs conforming the grid, as well as its access between the organizational departments, and services provided to third party organizations.

When receiving a service description (Web Services Description Language (WSDL)) and data from the service requester, the ESB is responsible for selecting the service which best fits to the description requirements, for binding the service requester with the backend service through a route created between the logical endpoints and for making the necessary data transformations to enable the communication between the parts.

As the ESB is the central component in SOA, and established as integration middleware for services, in this diploma thesis we focus on the required modifications and extensions in the open-source ESB Apache ServiceMix 4.3 to provide transparent communication support between the applications and its data located in on-premise databases, or migrated to off-premise data stores.

2.3. Analyse von Use Cases für die Dokumentähnlichkeitsbestimmung

One of the main decision variables for utilizing a Cloud computing environment are capital expenditures. The main goal of a Cloud consumer is to minimize its business costs when migrating to the Cloud. According to Chong, a SaaS solution benefits a Cloud customer with the following advantages [**?**]:

- The Cloud consumer does not directly purchase a software license, but a subscription to the software offered as a service by the Cloud infrastructure.
- More than half of the IT investments of a company are made in infrastructure and its maintenance. In a SaaS solution this responsibilities are mainly externalized to the Cloud provider.
- A Cloud computing environment is based on the utilization of its resources simultaneously by a large number of Cloud consumers. For example, a Cloud provider that offers a centrally-hosted software service to a large number of customers can serve all of them in a consolidated environment and lower the customer software subscription costs while maintaining or lowering the provider's infrastructure, administration and maintenance costs.
- The cost leverage in the software utilization allows the Cloud providers to focus not only on big enterprises capable of large IT budgets, but also on the small business that need access to IT solutions.

Multi-tenancy in a SaaS environment allows the Cloud providers to lower the cost per customer by optimizing the resources usage in the Cloud infrastructure. The software serves multiple tenants concurrently, who share the same code base and data storage systems. Chong and Carraro [**?**] define a well designed SaaS application as scalable, multi-tenant-efficient and configurable. With this design patterns, the SaaS model enables the provider to *catch the long tail*. Business softwares are becoming more complex and tend to demand an individual customer support and an increase of the computing and storage resources in the infrastructure. This fact leads to an increase in the infrastructure investment and maintenance costs. However, if the previous requirements are eliminated and the provider's infrastructure is scaled to combine and centralize customers' hardware and services requirements, the price reduction limit can be decreased and, in effect, allow a wide range of consumers to be able to access this services.

The reasons discussed above are also applicable in the DBaaS and STaaS models. Storage and retrieval of data involve high maintenance and management costs. The data management cost is estimated to be between 5 to 10 times higher than the data gain cost [?]. Furthermore, storing data on-premise does not only require storing and retrieving data, but also requires dealing with disaster recovery, Database Management System (DBMS), capacity planning, etc. Most of the organizations prefer to lead their investments to their local business applications rather than becoming a data management company [?]. Cloud storage providers offer a pay-per-use storage model, e.g. based on storage capacity or based on number of connections to the storage system, and ensure that the stored data will persist over time and its access through the network. However, security and confidentiality are the main constraints when moving private data to a shared public infrastructure.

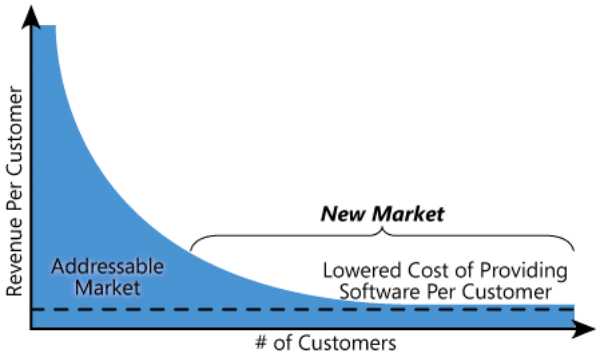

Figure 2.1.: New market opened by lower cost of SaaS [?]

In the Figure 2.1 the economics of scaling up to a high number of customers while reducing the software price is analyzed. Cloud providers have reached a new market formed by small or medium businesses without enough budget for building an on-premise IT infrastructure.

Multi-tenancy refers to the sharing of the whole technological stack (hardware, operating system, middleware, and application instances) at the same time by different tenants and their corresponding users [?]. Andrikopoulos et al. identify two fundamental aspects in multi-tenant awareness: communication, and administration and management [?]. The former involves isolated message exchanges between tenants and the latter allows tenants individual configuration and management of their communication endpoints. Utilizing an ESB as the central piece of communication middleware between applications in a PaaS environment forces it to ensure multi-tenancy at both communication, and administration and management, as mentioned before. The multi-tenancy support modifications made in the open-source ServiceMix 4.3 are the results of [?], [?], and [?]. In this diploma thesis we reuse and extend those results in oder to provide multi-tenant transparent Cloud data access in the Cloud through the ESB, when the application's data is migrated and accessed through the ESB in a Cloud infrastructure.

The migration of an application's stack to the Cloud can be done at different levels of the application's stack: Presentation Layer, Business Layer, and Data Access Layer. The Replacements of Components with Cloud offerings migration type is the least invasive type of migration [?].

In this diploma thesis we focus on this type of migration, concretely when the used Cloud offering is the database system. Migration of the data can be either seen as the migration of the Data Layer (Data Access Layer and Database Layer) or of the whole application [?]. Migration of the Data Layer to the Cloud means migrating the both data management and data access to the Cloud, while maintaining its transparency to the application's Business Layer.

In a Cloud infrastructure where Cloud storage is offered, Feresten identifies four main tenant requirements: security, performance, data protection and availability, and data management [?]. Multi-tenancy in a storage system can be achieved by aggregating tenant-aware metadata to the tenant's data (see Figure 2.2), or by physical storage partitioning, but this is not sufficient when fulfilling the data management, and the flexibility requirement. Tenants must have independent access and management, as if they accessed their own data storage systems. For this purpose, storage vendors have introduced the concept of *virtual storage container*, a tenant-aware management domain which grants all (or most of) the database management operations over the storage container, as described in Figure 2.2.

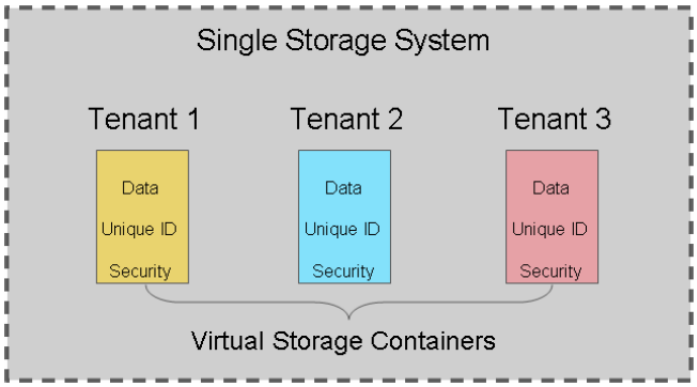

Figure 2.2.: Attributes of a Virtual Storage Container [?]

In this diploma thesis we must take into account the different approaches that most of the Cloud storage vendors have taken into account, in order to provide the tenant transparent access through the ESB to his virtual storage container in one or more Cloud infrastructures.

2.3.1. Finden von Dokumenten mit ähnlichen Inhalten(Duplikaten Findung)

2.3.2. Verwendung von a priori Wissen

2.3.3. Systemübergreifende „Fremdschlüssel"

2.3.4. Profil Matching

2.3.5. Email Klassifikation (Spamfilter)

2.4. Analyse existierender Ansätze zur Dokumentähnlichkeitsbestimmung basierend auf Bayesscher Statistik

2.4.1. More Like This

2.4.2. Naive Bayes

2.4.3. BayesLSH

The interaction between enterprises' applications has suffered in the past from lack of standardized technologies, leading each of the enterprises to develop their own or acquiring vendor-specific integration technology. Java Business Integration (JBI) is defined by the Java Community as an integration technology which maximizes the decoupling between components and defines an interoperation semantic founded on standards-based messaging. This allows different vendor-specific components to interoperate in a multivendor "echosystem" [**?**].

The key which leads to the integration of different components relies on a unique message format in the JBI environment which different plugged-in components use to communicate within the environment. External components are not directly connected, but through a mediator. The communication mediator between components in a JBI environment is the Normalized Message Router (NMR). Its main functionality is the routing of the internal standardized Normalized Message (NM) between the components. However, it can perform additional processing during the message exchange. The NMR fields are defined as an eXtensible Markup Language (XML) document format payload, metadata conforming the header and a non XML document format attachment referenced by the payload.

The JBI specification defines two different types of components which are categorized in two types and provide different services:

- A Service Engine (SE) provides transformation and composition services to other components.

- A Binding Component (BC) provides the connectivity between the external services and the JBI environment. They support many different protocols and isolate the JBI environment by marshaling and demarshaling the incoming or outgoing message into the internal standardized NM format.

Both components listed above can function as service consumers or service providers following a WSDL-based, service-oriented model. The consumer endpoint provides a service accessible through an endpoint which can be consumed by other components, while the provider endpoint consumes a functionality exposed as a service and accessible through an external endpoint. The routing of NM starts when a message exchange between components is created (bidirectional communication pipe, a *DeliveryChannel*, between the communicating endpoints) and continues with the target of the specified service endpoint for processing (see Figure 2.3). The NMR supports four asynchronous message exchange patterns differing in the reliability and direction of the communication.

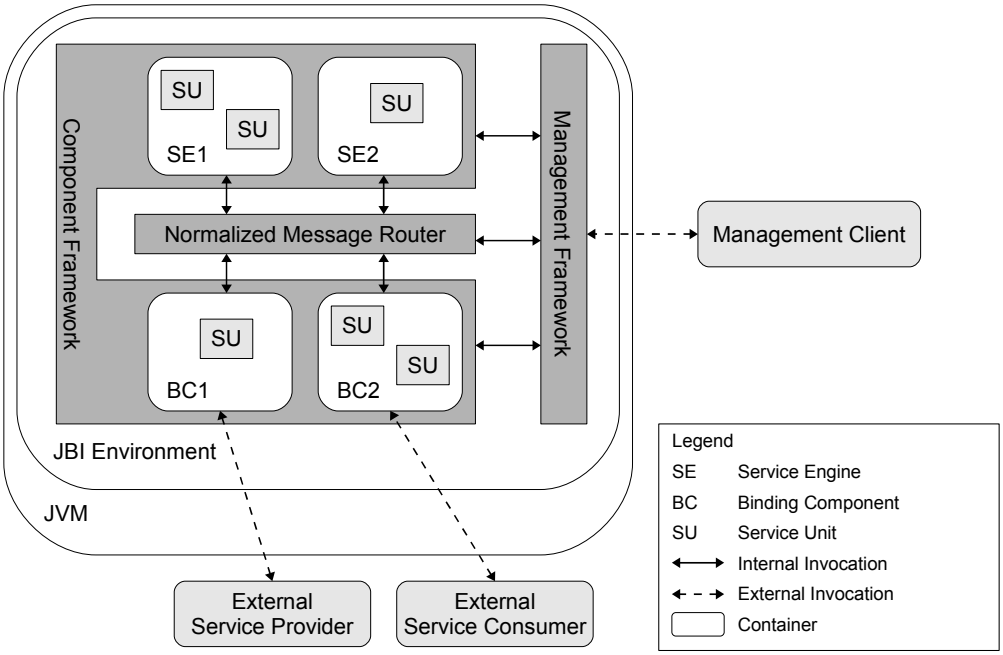

Figure 2.3.: Overview of JBI Architecture. Figure 4 in JBI specification document [**?**].

In Figure 2.3 we can observe that one or more Service Unit (SU) are contained in a BC. The SUs are component-specific artifacts to be installed to a SE or a BC [**?**]. The service units are packed in a Service Assembly (SA), usually as ZIP files, where it is specified each of the components where each of the SUs should be deployed. The JBI environment provides a Java Management Extension Java Management Extensions (JMX) Framework for installation, life cycle management, addition, and monitoring and control of the components conforming to the environment defined by the JBI specification.

3. Die Informationsrückgewinnung-Middleware-Lösung

In this chapter we provide a general overview on the different approaches that are taken into account in order to provide a reliable, secure, and transparent communication between on-premise application's layers and off-premise Cloud data stores. Furthermore, we discuss about the needed adaptations different authors specify that the on-premise application's layers must address when migrating their underlying layers to a Cloud infrastructure. We compare it to the ones we transparently support in our approach, and the ones the user should consider. We finally mention the improvements we need to perform to the original prototype ServiceMix-mt, and continue our discussion dividing it into the two main DBMS available nowadays in the market: SQL and Not only Structured Query Language (NoSQL) databases.

A migration process of the Database Layer of an application to the Cloud may pop up several incompatibilities with the new hosting environment, which need to be addressed prior to the migration decision. Strauch et al. aim to address such incompatibilities by defining a set of *Cloud Data Patterns*, which target finding a solution for a challenge related to the data layer of an application in the Cloud for a specific context [**?**]. Incompatibilities a user may find when migrating the application's data layer can be on the level of the schema representation, supported set of queries or query language, communication protocol, security, etc. Strauch et al. focus mainly in two non-functional aspects: enabling data store scalability and ensuring data confidentiality [**?**].

The former deals with maintaining the quality of service levels when the workload increases, for both write and read operations. There are two scalability mechanisms when dealing with data: vertical and horizontal scalability. A vertical scalable system can be obtained by introducing more powerful hardware, or moving to a more powerful database system, while a horizontal scalable system deals with splitting data into groups which are stored in different partitions or different database systems, also known as *sharding*. Due to the absence of support for accessing a *sharded database* between different database systems, the concepts of a database proxy and sharding-based router are introduced. In this first approach, a proxy component is locally added below each data access layer [**?**]. A proxy below each data access layer instead of a common proxy on top of the database layer dismisses a common point of failure when accessing the data. In the second approach, a local sharding-based router is added below each of the data access layer. A sharded-based router contains the needed knowledge about the location of the *sharded databases*. In our approach, we don't only partially follow both of the concepts, but integrate them in a single component. We consider a sharded-based router as a proxy with routing capabilities. Therefore, as it is discussed in Chapter **??**, enhancing an ESB with the required *sharded databases* knowledge and with standardized communication protocols, it allows us to utilize it as a sharded-based router, and as a proxy. Furthermore, the single point of failure avoidance can be ensured by increasing the number of ESB instances and balancing the load between them. As discussed before, we do not fully comply with this

approach. The development of a proxy or sharded-based router component below each data access layer forces each application to deploy at least one proxy or sharded-router instance in their system. In our approach we propose the utilization of our prototype as a shared transparent Cloud data access layer by connecting to a data access endpoint which supports a specific DBMS multi-tenant aware communication protocol (e.g. MySQL or PostgreSQL). For this purpose, we propose the concept of a lightweight Data Layer, where the adaptations to its sublayers are minimized, e.g. modification of data access host, port, etc. The data access endpoint acts as a database protocol-aware proxy, forwarding the requests to the NMR of the ESB. We enhance the Myosotis Tungsten Connector and provide access control, caching functionality, and full integration in the ESB OSGi container, and with the NMR [**?**].

Ensuring data confidentiality is presented in [**?**]. Their work deals with critical data management between on-premise and off-premise data stores, and categorizes data into different confidentiality levels to prevent data disclosures. The former is achieved by aggregating information which categorizes data into different categories and confidentiality levels. The latter deals with keeping confidential data on-premise. With data filtering, pseudonymization, and anonymization, data is either prevented from being externally routed, or secured when routed to a public Cloud [**?**]. The pseudonymization technique provides to the exterior a masked version of the data while maintaining its relation with the original data, and the anonymization provides to the exterior a reduced version of the data. In this diploma thesis' approach, we assume that the application's owner has decided on which data should be and cannot be migrated, and that the business layer is hosted on-premise. Therefore, there is no data processing in a public Cloud environment. Our final prototype provides confidentiality between different tenants of the system by injecting tenant information in our messages and providing tenant-aware routing, and different multi-tenant aware endpoints. We do not need to provide support for pseudonymization or anonymization techniques, in contrast to [**?**].

Replacement of components which build an application with Cloud offerings leads the developers to face an application's adaptation process. For example, migrating a local database to a private Cloud or to a public Cloud, or sharding a database between on-premise and off-premise data stores forming a single data store system, can not be accessible without adapting the non-migrated application's layers to the new storage system. Andrikopoulos et al. identify the needed adaptations actions when migrating a data layer to the Cloud [**?**]: address reconfiguration, patterns realization, incompatibilities resolution, query transformation, and interaction with data store allowance. Our main goal in our final prototype is to minimize the number of adaptations the user must perform when migrating application's data to a Cloud data store. The adaptations of the ESB must encompass the described adaptations in a transparent way to the user, in order to internally support in our prototype compatibility between the application and the different data stores, and lower the adaptation operations number at the application's side, e.g. only address reconfiguration.

Federated Database Systems are a type of Multidatabase System (MDBS) that allow accessing and storing data which is stored in different and noncontiguous databases through a single interface. Sheth and Larson define them as a collection of cooperating but autonomous component database systems, which can be integrated to various degrees, and can be accessed by a software which controls and coordinates the manipulation of the database systems which

conforming the federated database system. This distributed database model allows users to access and store data among different database systems, which can be located in different continents, without dealing with multiple connections to the different database systems, query and data transformation, address adaptation, etc. MDBS are accessed through a single endpoint which provides a single logical view of the MDBS, and users can access the different DBMS which form the MDBS (see Figure 3.1).

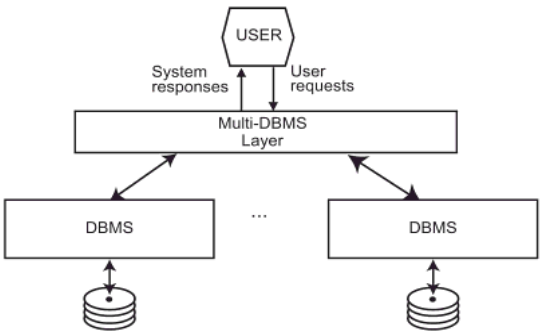

Figure 3.1.: Components in a multidatabase system [**?**]

A popular implementation architecture for a MDBS is the mediator/wrapper approach [**?**]. Mediators exploits knowledge to create information for upper layers, while wrappers provide mapping between different views, e.g. relational vs. object-oriented views. We can consider our approach as a MDBS with some modifications and less functionalities. In the first place, using the ESB as a single entrance point to the data system while managing different backend autonomous Cloud or traditional data stores comply with the main concept of a MDBS. Furthermore, Cloud data store providers may implement the same distributed database model, whereby we could find two logical levels for accessing the physical data. However, we do not accurately follow the mediator/wrapper approach. In our approach we exploit data provided by the tenant during the migration decision and process, by providing an interface to register the backend data store/s information in our system, for future routing purposes. Furthermore, compatibility information is registered in order to apply the needed query or data transformation between data stores. However, the transformation is out of the scope of this diploma thesis, and the support of table joins between databases located in different Cloud data stores are out of scope as well.

As described in the previous chapter, multi-tenancy is one of the main requirements in a Cloud environment. Muhler, Essl, and Gomez provide an extended version of ServiceMix 4.3, which supports multi-tenancy at two different levels: communication, and administration and management [**?**], [**?**], [**?**]. However, their prototype supports tenant isolation at the level of tenants. A DBMS, e.g. MySQL, by default provides access to one default user and supports multiple users creation [**?**]. Therefore, in our approach we must not only consider isolation at the tenant level, but also at the user level. We assume that the tenant is the default user which migrates his data store to a Cloud environment, but the migrated data store may contain one or more users. In our prototype we ensure tenant and user isolation at both communication, and administration and management levels.

Over the past decades, caching has become the key technology in bridging the performance gap across memory hierarchies via temporal or spatial localities; in particular, the effect is prominent in disk storage systems [**?**]. Han et al. investigate how cost efficiency in a Cloud environment can be achieved, specially in applications which require a high I/O activities number, and present a CaaS (cache-as-a-service) model. Cloud providers offering data storage solutions present pricing models based on the storage size, usage per time, or number of requests. Amazon RDS costs $0.025 per hour for a Micro DB Instance usage [**?**], while Amazon DynamoDB $0.01 per hour for every 50 units of read capacity [**?**], and Google Cloud Storage $0.01 per 1000 PUT, POST, GET requests per month [**?**]. An I/O-intensive application whose database is hosted in the Cloud may produce a significant economic cost. The cost of continuously retrieving data from the Cloud data store, when existing temporal proximity between the data accessed, can be considered unnecessary, and reducible. Furthermore, the application's overall performance can be reduced due to the network latency and, in the scope of this work, the use of an ESB to access the Cloud data store. In this diploma thesis we do not provide cashing as a service, but include cashing support to the sharded-based router pattern described in [**?**]. Uralov enhaces ServiceMix-mt with cashing support for dynamic discovery and selection of Cloud data hosting solutions [**?**]. However, we must adapt and extend it due to the lack of support of functionalities we require and the lack of full OSGi compliance.

3.1. Vorstellung

In this section we describe the JBI BCs shipped in the ServiceMix-mt prototype this diploma thesis focuses on, and the transport protocols they support.

ServiceMix provides HTTP communication support in its HTTP JBI BC. Its HTTP consumer and provider endpoints are built on the HTTP Jetty 6 server and Jakarta Commons HTTP Client respectively, providing support for both REST and SOAP over HTTP 1.1 and 1.2 requests.

The original HTTP BC is extended in the ServiceMix-mt prototype to provide multi-tenant support in [**?**] and [**?**]. Muhler provides an internal dynamic creation of tenant-aware endpoints in the BC, by injecting tenant context in the JBI endpoint's URLs [**?**]. Gomez provides a Normalized Message Format (NMF) with tenant context information in its properties for routing in the NMR [**?**]. However, in this diploma thesis we must not only provide tenant isolation at the tenant level, but also isolation at the user level. We discuss this requirement in detail in Chapters **??** and **??**.

As seen in Figure 3.2, the multi-tenant HTTP BC is mainly used in ServiceMix-mt to support the SOAP over HTTP communication protocol by exposing a Web service in the tenant-aware consumer endpoint and consuming an external Web service in the provider endpoint. SOAP defines an XML message format which is sent over the network and a set of rules for processing the SOAP message in the different SOAP nodes which build the message path between two endpoints [**?**]. A SOAP message is a composition of three main elements: a

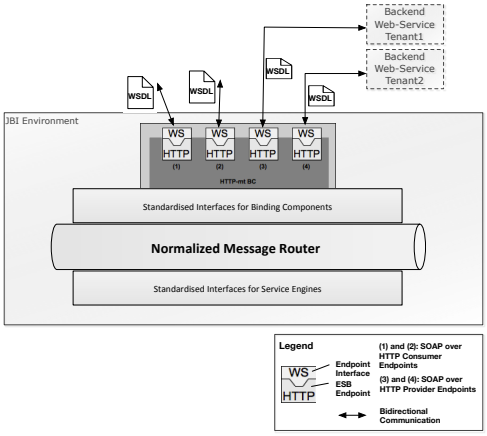

Figure 3.2.: Multi-tenant HTTP Binding Component [**?**].

SOAP envelope, header, and body. A SOAP envelope may contain zero or more headers and one body. The header may contain processing or authentication data for the ultimate receiver or for the intermediate nodes through the message is routed. The message payload or business data is included in the SOAP body. SOAP is used as a message framework for accessing Web services in loosely coupled infrastructures [**?**]. The Web service consumer specifies the functionality to invoke in the SOAP body. If the Web service functionality has a request-response Message Exchange Patterns (MEP), a SOAP message is used to send the response data when the corresponding operation has been executed successfully or the error data in case an error occurred during execution.

Most of the Cloud storage providers provide an HTTP interface to the tenants for data management, retrieval, and storage. In this diploma thesis we extend this JBI BC in order to provide the tenant a transparent access to his NoSQL Cloud data stores.

3.2. Anforderungen

The Java Naming and Directory Interface (JNDI) defines a framework for deployment support in a Java Virtual Machine (JVM) of downloaded or extended applications known as *bundles*. This framework requires OSGi-friendly devices a minimum system's resources usage by providing dynamic code-loading and *bundle* lifecycle management. An OSGi *bundle* is the packaging of a group of Java classes and required and provided capabilities' meta-data as a JAR file for providing functionality to end users. OSGi *bundles* can be downloaded, extended and installed remotely or locally in the platform when needed without the need of system reboot. Installation and update of bundles during their lifecycle are also managed by the framework, which uses a service registration for selection, update notifications, or registry of new service objects offered by a deployed bundle. This feature is the main key for connecting bundles whose's services require during runtime capabilities provided by another bundles. The framework defines a bundle's requirement capability as a dependency.

The OSGi framework defines 5 different layers and a bundle's lifecycle [**?**]. An optional Security Layer provides the infrastructure for deploying and managing applications which must be controlled during runtime. The Module Layer lists the rules for package sharing between the deployed bundles. The lifecycle of a bundle can be modified during runtime through an API provided in the lifecycle layer. The main operations implemented are install, update, start, stop or uninstall.

3.2.1. Funktionelle Anforderungen

3.2.2. Nicht funktionelle Anforderungen

3.2.3. Plattformanforderungen

3.2.4. Komponentenanforderungen

3.2.5. Anforderungen an dem Ähnlichkeitsalgorithmus

3.3. High-Level Architektur

A multi-tenant management system must fulfill several requirements, such as data and performance isolation between tenants and users, authentication, specification of different user roles, resources usage monitoring, etc. In a JBI environment, endpoint and routing configurations files are packed in SUs, and the latter in SAs for deployment. However, there is a lack of user-specific data during deployment. Muhler solves this problem in JBIMulti2 by injecting tenant context in the SA packages, making them tenant-aware [**?**].

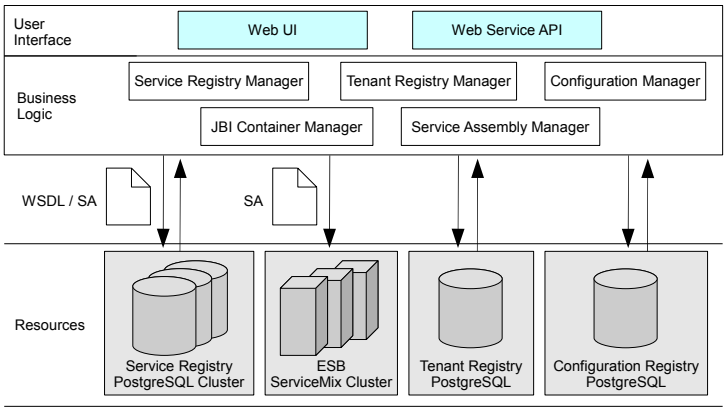

Figure 3.3.: JBIMulti2 System Overview [**?**]

The architecture of the JBIMulti2 system is represented in Figure 3.3. We can distinguish two main parts in the system: business logic and resources. JBIMulti2 uses three registries for storing configuration and management data. When a tenant (or a tenant user) is registered, an unique identification number is given to them and stored in the Tenant Registry. Both Tenant Registry and Service Registry are designed for storing data of more than one deployed application. The former for storing tenant information and the latter for providing a dynamic service discovery functionality between the different applications accessed through the ESB. The Configuration Registry is the key of the tenant isolation requirement of the system. Each of the stored tables are indexed by the tenant id and user id value. In this thesis we need tenant information during runtime. We reuse and extend the databases schemas produced by Muhler, specifically the Service Registry.

The system provides a user interface for accessing the application's business logic. Through the business logic, the management of tenants can be done by the system administrator or the management of tenant's users can be done by the tenants. Furthermore, when deploying the different tenant's endpoint configurations packed in SAs, the system first makes modifications in the zip file for adding tenant context information and then communicates with the Apache ServiceMix instance by using a Java Message Service (JMS) Topic to which all the ServiceMix instances are subscribed to. The JMS management service in ServiceMix deploys the received SA injected in the received JMS message using the administration functionalities provided in ServiceMix. The communication between the business layer and the ServiceMix instance is unidirectional. When successful deployment, the endpoint is reachable by the tenant. When an error occur during deployment, an unprocessed management message is posted in a dead letter queue.

JBIMulti2 requires the previous installation of components, e.g. JOnAS server, PostgreSQL, etc. The initialization of the application is described in both Chapter **??** and in the JBIMulti2 setup document [**?**].

4. Auswahl von Ansätzen zur Dokumentähnlichkeitsbestimmung in der Information-Retrieval-Middleware-Lösung

In this chapter we first provide an overview of the system and its components which provide support for accessing on-premise and off-premise Cloud data stores after migrating the data to the Cloud. In the second part of this chapter we specify the functional and non-functional requirement the system must fulfill, and provide a list of the use cases, which extend the use cases description provided in [**?**] and [**?**].

4.1. Kriterien der Auswahl

To provide transparent access support for migrated data, we present in this section an overview of the system, and its components. As we can see in Figure 4.1, we divide the system into two main parts: the *Cloud Data Migration Application*, and the Cloud data access subsystem, which we name in this diploma thesis CDASMix (Cloud Data Access Support in ServiceMix-mt). However, in this diploma thesis we do not focus on the *Cloud Data Migration Application*, but include it in the system's overview in order to explain the role of CDASMix in the context of the migration of the DL to the Cloud. We consider the different tenant's applications hosted in their environment not as part of our system, but as consumers of the services provided by it. We must specify that the system overview described in Figures 4.1 and 4.2 shows the state after the data migration, when the data is already hosted in the backend Cloud provider. However, we include the migration process explanation in this section.

In the first part of our system, the *Cloud Data Migration Application* provides support for the data migration process, from an traditional to a Cloud data store, or between Cloud data stores [**?**]. After the tenant provides the required source and target data store configuration, the application calculates possible incompatibilities between data sources, and presents them to the tenant. If they exist, the tenant must resolve the incompatibilities before migrating the data. In the end phase of the migration process data can be easily migrated to the Cloud by providing the application with the DBMS access credentials.

From the point in time where the data migration process is terminated, either the application or the tenant must choose if he directly connects to his data source in the Cloud, or if he prefers to transparently access his data in the Cloud utilizing our Cloud-enabled data bus. If the latter is chosen, either the application or the tenant must register which communication protocol is required and register access and configuration data in our registry, e.g. database type, database URL, access credentials, etc. For this purpose, we enhance the administration

and management system's (JBIMulti2) Web service Application Programming Interface (API) with Cloud data access registering capabilities, as described in the following sections.

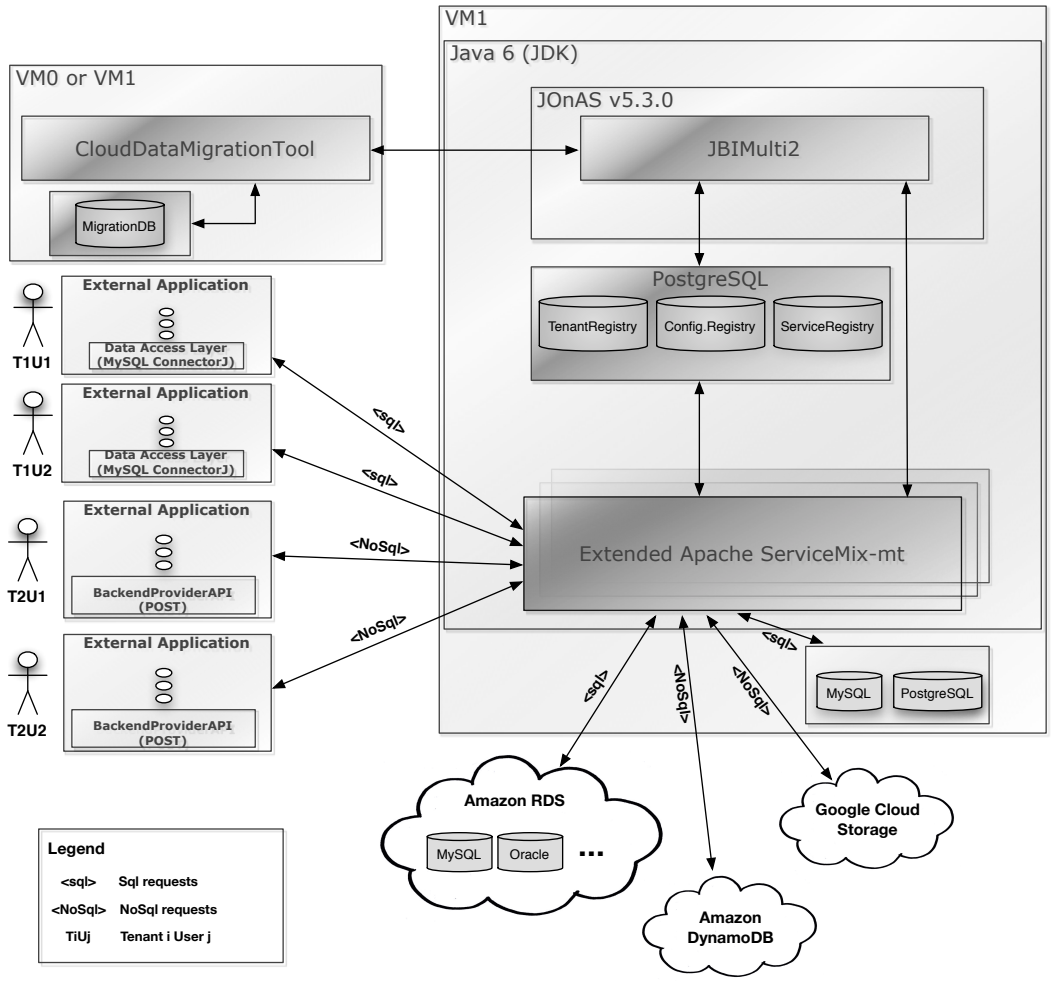

Figure 4.1.: Transparent Cloud data access system overview, including the *Cloud Data Migration Application* [**?**]. Note: represents the system in the post-migration phase

The transparent Cloud data access access support is achieved by the interaction of three main components: JBIMulti2, registries containing tenant-aware information, and an extended version of ServiceMix-mt (see Figure 4.1). JBIMulti2 deploys in ServiceMix-mt the SAs containing the endpoint and routing configurations selected by the tenant, which support two different communication protocols: MySQL and HTTP. From this point the DAL of the tenant's application can retrieve and modify data in his data container in the Cloud through the ESB connecting to a single logical endpoint which connects to multiple physical backend data stores. In our approach we provide also the possibility, either to configure a connection to the traditional database, e.g. when a hybrid model is pursued, or to utilize a DBMS provided in our system, which is described in the following subsection.

4.1.1. Funktionsweise des Ansatzes

In Figure 4.2 we specify the main components which build the subsystem mentioned in Section **??**. We highlight the components which require an extension, and the new components which are implemented and included in the system.

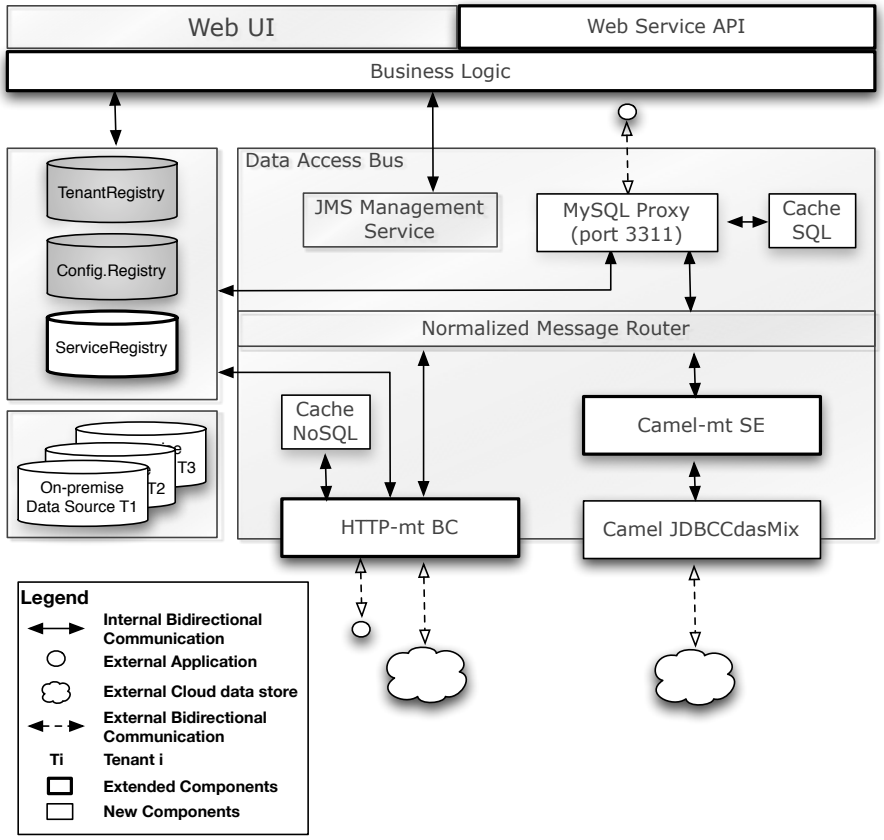

Figure 4.2.: Transparent Cloud data access components overview.

As described in Section **??**, we extend the JBIMulti2 Web service API and its associated business logic. The operations we include perform access and modifications to one particular registry: the Service Registry. This registry persists information about services, its policies, and SAs deployed by one tenant. The last kind of information is the one we mainly focus on, due to the information which is contained in it: the tenant-aware endpoint configuration in ServiceMix-mt. Therefore, we extend this registry to persist the configuration about the data stores. We make a differentiation between data stores and name it source and target data sources, to be able to relate the one that the tenant physically accesses and the one which the tenant logically accesses, which is the one that the system physically accesses. To support transparent access to two different database types, we divide our architecture and implementation into the the communication protocols they support: MySQL for MySQL databases, and HTTP for NoSQL databases (see Figure 4.2).

For the first one, the single access physical endpoint is provided through a Transmission

Control Protocol (TCP) port, which then forwards the message to the appropriate tenant's endpoint in the multi-tenant Servicemix Camel component, and this to the component which physically connects to the backend SQL DBMS. For the second one, we extend the Servicemix-http-mt in order to physically connect to the backend NoSQL data stores.

The possibility of migrating a database to the VM instance where the ESB runs is also supported. However, we do not provide a multi-tenant DBMS where a database or table is shared between multiple tenants, since it is not a requirement in this diploma thesis. The ensured isolation in this case is the one provided by a DBMS between different databases. Furthermore, backup, restoration, administration, and horizontal scalability services are not supported. We provide in the VM instance where CDASMix runs a MySQL database system. The number and types of databases systems supported in the VM instance relies on the administrator, and the PostgreSQL database system where the system registries are stored must be independent from the PostgreSQL instance which hosts the migrated tenant's databases.

As represented in Figure 4.2, we enhance ServiceMix-mt with cashing support, in particular for the two different types of databases we support. The cashing mechanism supports storage of access control data, as well as retrieved data from the backend data store, e.g. a bucket, or a set of rows. The reasons for using a divided cashing system instead of a single one is explained in Chapter **??**.

5. Implementierung einer Teilmenge ausgewählten Ansätzen

In this chapter we present the architectural solution taken into account to build the system which fulfills the requirements specified in Chapter **??**. Due to the required communication support for SQL and NoSQL databases, we separate the architectural approaches and provide them separately. JBIMulti2 and ServiceMix-mt are the subsystems we must reengineer in order to aggregate transparent and dynamic routing functionalities. Therefore, we also provide in this chapter the needed extensions in the components conforming the system, e.g. service registry in JBIMulti2, and NMF in ServiceMix-mt.

6. Evaluierung der Skalierbarkeit von den realisierten Ansätzen basierend auf ausgewählten Probendaten

In this chapter we describe the challenges and problems during the implementation phase to fulfill the requirements specified in Chapter **??** and the design presented in Chapter **??** of the system. Furthermore, we discuss the incompatibilities found with components we must extend. We divide, as in the previous chapters, the implementation phase into the SQL and NoSQL databases support, and provide a separate section for the extensions made to JBIMulti2 and the Cache.

7. zusammenfassung und Ausblick

In this chapter we provide the validation, and evaluation of the system. We must ensure that the requirements specified in Chapter **??** are fulfilled in the design and implementation phases. In Section **??** we describe the steps which should be followed to initialize the system, and the testing scenarios. After the initialization we execute the test cases in Section **??**, and monitor the incoming requests to ServiceMix-mt, and the outgoing requests to the backend Cloud data store. Due to the extensions implemented on the ESB, we evaluate in Section **??** its behavior, and the impact that our modifications have on the original ServiceMix-mt.

Appendix A.

Components

A.1. CDASMix MySQL Proxy

The MySQL proxy OSGi bundle is implemented on the Continuent Tungsten Connector [**?**], which is a Java MySQL proxy which directly connects with the backend MySQL database system. We extend and adapt this proxy in order to integrate it with ServiceMix, aggregate transparency, multi-tenant awareness, cashing, and dynamic connection with the backend Cloud data sources.

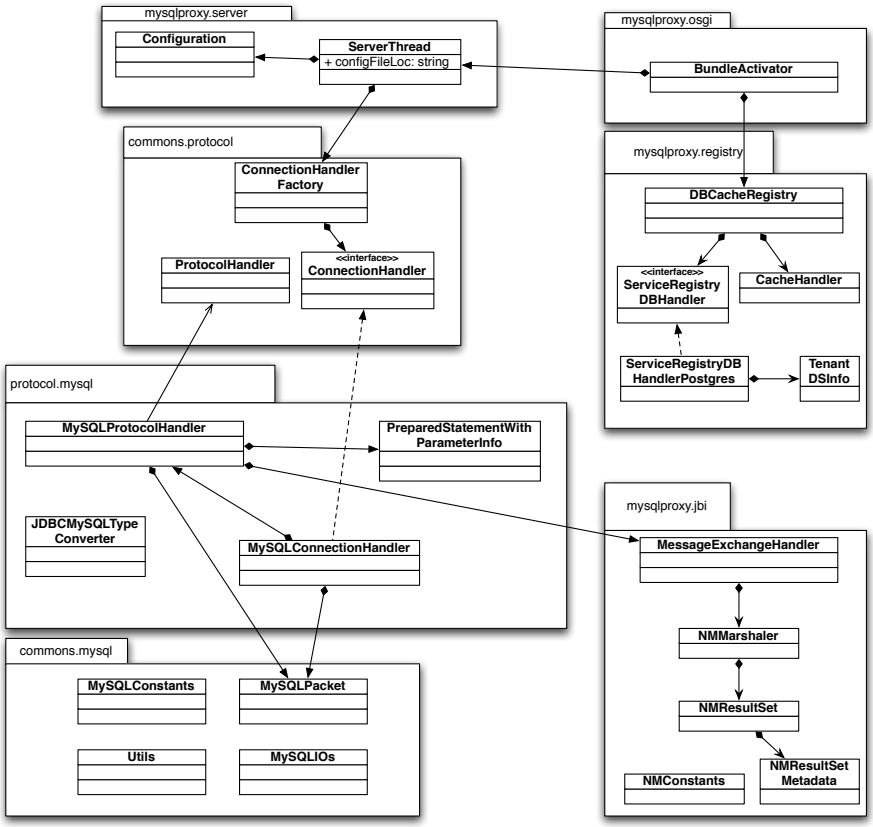

Figure A.1.: OSGi bundle providing MySQL support in ServiceMix-mt

A.2. CDASMix Camel JDBC

The *cdasmixjdbc* component is a custom component which is built and deployed as an OSGi bundle in ServiceMix-mt. It provides support for connections with backend SQL Cloud data stores, and message marshaling and demarshaling.

Figure A.2.: OSGi bundle and Camel component providing JDBC support in ServiceMix-mt

Appendix B.

Messages

In this chapter we provide an overview of the requests which are sent to, and received from the extended ServiceMix-mt. For MySQL requests we provide the TCP packets which are transferred between the application, ServiceMix-mt, and the backend MySQL database system. For NoSQL requests we present messages samples which are in JavaScript Object Notation (JSON) format, but its content varies among the different backend Cloud data store providers.

B.1. Normalized Message Format Content Description

In this section we provide an overview of the data structures which are sent in the NMF. The sections of the NMF where the data and meta-data are sent are the *properties*, and the *attachment*. In the Listing B.1 we detail the contents sent in each of the sections, and the data structures in which the data and meta-data are stored.

```
1  ########## Normalized Message Adaptation for CDASMix ############
2
3  Properties for the backend connections are stored in this format. At this time, only joins
       which involve tables in the same backend db are supported:
4
5  ////////////////////////////////////////////////////////////////////
6
7    - MySQL || NoSql -> NMF (Request Message)
8      // Main properties
9      - target_data_sources : number of target data sources. This number will set the length
           of the properties vector and will set the length of the vector queries
10     - tenantId : string (UUID)
11     - userId : string (UUID)
12     if (mysql)
13       - mysqlServerPropsQueries:vector<string>      // server configuration queries that the
             proxy has received from the jdbc driver, e.g. SET names, SET character_set_*
14
15     // Backend datasource dependent properties : stored as Properties in vector<properties>.
16     - Tenant configuration data
17       - source & target dataSource name : string
18       - source & target dataSource type : family-mainInfoStructure-secondaryInfoStructure-
             version
19       - source & target dataSource protocol : mysql | http.[xml|rest]
20       - target datasource endpoint url: string
```

```
21        - target datasource user and password : string, string
22        - source & target endpoint type: endpoint.type.jdbc | endpoint.type.http | endpoint.
              type.jms
23        - targetJBIEndpointURI: QName
24        - source and target information structure information:
25               if source is sql
26                 - src_main_info_structure_name : string
27                 - target_main_info_structure_name : string
28               if source is nosql
29                 - src_main_info_structure_name : string
30                 - target_main_info_structure_name : string
31                 - src_secondary_info_structure_name : string
32                 - target_secondary_info_structure_name : string
33
34        - if the target protocol == mysql
35          - target native driver name : native driver name, e.g. "com.mysql.jdbc.driver"
36          - bean DS name (if exists)
37          - escapeProcessing : true | false               //statement property of mysql
38          - fetchSize : int (statement property of mysql)
39          - returnGeneratedKeys : true | false            //statement property of mysql
40
41      // Attachment
42      if (SQL)
43        - Query/ies go/es in the NMF body as a vector<String>, with query data (for insert
              queries).
44      else
45        - NoSQL JSON payload
46
47
48  ////////////////////////////////////////////////////////////////////////
49
50    - NMF -> MySQL || NoSql (Response Message)
51      // Main properties
52      - target_data_sources : number of target data sources. This number will set the length
            of the properties vector and the length of the result
53      - tenantId : string (UUID)
54      - userId : string (UUID)
55      - target_op_status = ok | error
56
57      - if source protocol == mysql
58
59        - updateCount = int           // the current result as an update count
60        - resultSetMetadataNumber: int      // number of result set in the vector of response
              (index in the result set metadata vector)
61
62      // Backend datasource dependent properties : stored as Properties in the vector<
            properties>.
63      - source & target dataSource name : string
64      - source & target dataSource type : family-mainInfoStructure-secondaryInfoStructure-
            version
65      - source & target dataSource protocol : mysql | http.[xml|rest]
66      - source & target endpoint type: endpoint.type.jdbc | endpoint.type.http | endpoint.type
            .jms
67      - source and target information structure information:
68             if source is sql
```

```
69              - src_main_info_structure_name : string
70              - target_main_info_structure_name : string
71            if source is nosql
72              - src_main_info_structure_name : string
73              - target_main_info_structure_name : string
74              - src_secondary_info_structure_name : string
75              - target_secondary_info_structure_name : string
76
77    // Attachment
78    - if target_op_status == error
79      - target_op_error : vector<Hashmap<String,String>>            // map of error code and
            error message per backend db
80      else
81
82
83      - if source protocol == mysql
84
85        - vector<arraylist[HashMap<columnName,value>]>           // result sets are
            inserted as arraylist in the body
86
87        - ResultSetMetadata: Vector<HashMap<String,Object>>
88          - columncount : int                       // number of columns in this ResultSet
              object.
89          - columntype : arraylist<int>                 // the designated column's SQL type.
90          - isnullable : arraylist<int>                 // if the column can contain null
              values
91          - isAutoIncrement : arraylist<int>               // if the column values
              increment when the rows do
92          - tableName : arraylist<string>                 // name of the table where a
              column is contained
93          - stringcolumnlabel : arraylist<string>             // label which sql proposes
              for printing reasons
94          - stringcolumnname : arraylist<string>              // column name
95          - ColumnDisplaySize : arraylist<int>              // Indicates the designated
              column's normal maximum width in characters.
96          - Scale : arraylist<int>                    // column's number of digits to right
              of the decimal point
97          - RowCount : int                          // number of rows
98          - ColumnMapper : Hashmap<int,String> (column index, name)   // mapping of the
              column index with the column name
99          - ColumnSigned : arraylist<int>                 // if the values of the columns
              are signed or unsigned
100       else
101       - versionControl
102       - nosql metadata
103       - nosql JSON payload
104
105 ////////////////////////////////////////////////////////////////////
106
107   Note: response to the user -> distributed atomic transaction mechanism used. If one of
        the backend datasources reports an error as a response, the final response to the
        tenant is an error response:
108
109     - Error response: string of the error + the datasource/s which gave the error.
110
```

```
111       - OK response: the full result set, this means, the vector<arraylist[HashMap<
              columnName,value>]]> created in the data transformation SE or jdbccdasmix
              component.
112
113
114   ////////////////////////////////////////////////////////////////////////
```

Listing B.1: Detail of the content and data structures used to send the requests' data and meta-data.

B.2. MySQL TCP Stream

In this section we provide two TCP streams which are captured with the *ngrep* program for UNIX [**?**]. The first stream captures the TCP packets on port 3311, where the MySQL component in ServiceMix-mt listens for incoming connections (see Listing B.2). The second stream captures the TCP packets on port 3306, where the a locally deployed MySQL server listens for incoming connections (see Listing B.3).

```
interface: eth3 (109.231.70.232/255.255.255.248)
filter: (ip or ip6) and ( port 3311 )

T 109.231.70.234:3311 -> 129.69.214.249:52190 [AP]
  45 00 00 00 0a 35 2e 31    2e 31 2d 53 65 72 76 69    E....5.1.1-Servi
  63 65 4d 69 78 2d 34 2e    33 2e 30 00 2d 02 00 00    ceMix-4.3.0.-...
  53 28 72 4d 51 30 6c 61    00 00 0d a2 02 00 00 00    S(rMQ0la........
  00 00 00 00 00 00 00 00    00 00 00 32 2f 54 6f 44    ...........2/ToD
  4d 4a 39 2a 70 69 49 00    00                         MJ9*piI..

  ...

T 129.69.214.249:52190 -> 109.231.70.234:3311 [AP]
  0f 00 00 00 03 53 45 54    20 4e 41 4d 45 53 20 75    .....SET NAMES u
  74 66 38                                              tf8

T 109.231.70.234:3311 -> 129.69.214.249:52190 [AP]
  07 00 00 01 00 00 00 02    00 00 00                   ...........

  ...

T 129.69.214.249:52190 -> 109.231.70.234:3311 [AP]
  1d 00 00 00 03 73 65 6c    65 63 74 20 2a 20 66 72    .....select * fr
  6f 6d 20 6d 61 69 6e 49    6e 66 6f 54 65 73 74 31    om mainInfoTest1
  3b                                                    ;

T 109.231.70.234:3311 -> 129.69.214.249:52190 [AP]
  01 00 00 01 02 46 00 00    02 03 64 65 66 12 69 6e    .....F....def.in
  66 6f 72 6d 61 74 69 6f    6e 5f 73 63 68 65 6d 61    formation_schema
  0d 6d 61 69 6e 49 6e 66    6f 54 65 73 74 31 0d 6d    .mainInfoTest1.m
  61 69 6e 49 6e 66 6f 54    65 73 74 31 02 49 44 02    ainInfoTest1.ID.
  49 44 0c 21 00 0b 00 00    00 03 01 02 00 00 00 4a    ID.!...........J
  00 00 03 03 64 65 66 12    69 6e 66 6f 72 6d 61 74    ....def.informat
  69 6f 6e 5f 73 63 68 65    6d 61 0d 6d 61 69 6e 49    ion_schema.mainI
  6e 66 6f 54 65 73 74 31    0d 6d 61 69 6e 49 6e 66    nfoTest1.mainInf
  6f 54 65 73 74 31 04 6e    61 6d 65 04 6e 61 6d 65    oTest1.name.name
  0c 21 00 ff ff 00 00 fc    01 00 00 00 00 05 00 00    .!..............
  04 fe 00 00 02 00 13 00    00 05 01 33 10 54 68 69    ...........3.Thi
  73 69 73 41 4e 65 77 4e    61 6d 65 33 33 08 00 00    sisANewName33...

  ...

T 129.69.214.249:52190 -> 109.231.70.234:3311 [AP]
  01 00 00 00 01                                        .....

T 109.231.70.234:3311 -> 129.69.214.249:52190 [AP]
  07 00 00 01 00 00 00 02    00 00 00                   ...........

T 129.69.214.249:52190 -> 109.231.70.234:3311 [R]
  00 00 00 00 00 00                                     ......

T 129.69.214.249:52190 -> 109.231.70.234:3311 [R]
  00 00 00 00 00 00                                     ......
```

Listing B.2: TCP Stream for a MySQL communication captured on port 3311 with the program *ngrep* [**?**].

```
interface: lo (127.0.0.0/255.0.0.0)
filter: (ip or ip6) and ( port 3306 )

  ...

T 127.0.0.1:46409 -> 127.0.0.1:3306 [AP]
  0f 00 00 00 03 53 45 54    20 4e 41 4d 45 53 20 75    .....SET NAMES u
  74 66 38                                              tf8

T 127.0.0.1:3306 -> 127.0.0.1:46409 [AP]
  07 00 00 01 00 00 00 02    00 00 00                   ...........

  ...

T 127.0.0.1:46409 -> 127.0.0.1:3306 [AP]
  50 00 00 00 03 75 70 64    61 74 65 20 6d 61 69 6e    P....update main
  49 6e 66 6f 54 65 73 74    31 20 73 65 74 20 6e 61    InfoTest1 set na
  6d 65 3d 27 54 68 69 73    69 73 41 4e 65 77 4e 61    me='ThisisANewNa
  6d 65 33 33 27 20 77 68    65 72 65 20 6e 61 6d 65    me33' where name
  3d 27 54 68 69 73 69 73    41 4e 65 77 4e 61 6d 65    ='ThisisANewName
  32 32 27 3b                                           22';

T 127.0.0.1:3306 -> 127.0.0.1:46410 [AP]
  45 00 00 00 0a 35 2e 31    2e 36 37 2d 30 75 62 75    E....5.1.67-0ubu
  6e 74 75 30 2e 31 30 2e    30 34 2e 31 00 c0 02 00    ntu0.10.04.1....
  00 5f 67 6b 63 2f 5e 6a    7b 00 ff f7 08 02 00 00    ._gkc/^j{.......
  00 00 00 00 00 00 00 00    00 00 00 00 4e 53 35 4e    ............NS5N
  68 44 57 6a 2f 48 5e 21    00                         hDWj/H^!.

T 127.0.0.1:46410 -> 127.0.0.1:3306 [AP]
  4a 00 00 01 8f a2 02 00    ff ff ff 00 21 00 00 00    J...........!...
  00 00 00 00 00 00 00 00    00 00 00 00 00 00 00 00    ................
  00 00 00 00 72 6f 6f 74    00 14 f3 54 d0 fa f3 56    ....root...T...V
  e9 c0 43 2c 4c 78 18 88    ac de 4c 5e aa d1 64 61    ..C,Lx....L^..da
  74 61 53 6f 75 72 63 65    54 65 73 74 31 00          taSourceTest1.

T 127.0.0.1:3306 -> 127.0.0.1:46410 [AP]
  07 00 00 02 00 00 00 02    00 00 00                   ...........

  ...

T 127.0.0.1:46410 -> 127.0.0.1:3306 [AP]
  1d 00 00 00 03 73 65 6c    65 63 74 20 2a 20 66 72    .....select * fr
  6f 6d 20 6d 61 69 6e 49    6e 66 6f 54 65 73 74 31    om mainInfoTest1
  3b                                                    ;
```

Listing B.3: TCP Stream for a MySQL communication captured on port 3306 with the program *ngrep* [**?**].

Bibliography

All links were last followed on March 21, 2013.

Acknowledgement

I am heartily thankful to my supervisor Steve Strauch from the University of Stuttgart for his encouragement, guidance and support in all the phases of this diploma thesis. I am also grateful to Dr. Vasilios Andrikopoulos for his advices and useful tips. Special thanks to my family, friends and girlfriend for their moral support.

Santiago Gómez Sáez

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

Stuttgart, 22nd March 2013 _____

(Santiago Gómez Sáez)