Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masters thesis

A framework for service-based data processing

Khaled Mahrous

Course of Study:	Computer Science
Examiner:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Supervisor:	Dipl.-Inf. Pascal Hirmer
Commenced:	28. February 2017
Completed:	28. August 2017
CR-Classification:	I.1.4, I.6.5, J.0

Abstract

The amount of data generated everyday in IT environments keeps increasing. In order to be able to make use of the large quantities of data generated in actual practical applications, complex computational requirements are needed to process and understand the data. It is expected that the amount of data created in the future will increase exponentially. As a result, the processing of data, which includes the extraction, transformation, and analysis of it, will also become more complex over time. For this Masters thesis, a framework for service-based data processing was developed by improving upon the application FlexMash developed at the University of Stuttgart for data integration. The improvements were done by using Business Process Model and Notation (BPMN), and creating a Service Platform for data processing services, in order to be able to decouple the services offered from the application itself. Furthermore, the improvements provide a simple way to integrate new services to the application in a generic manner, and enable an ad-hoc approach for data processing services. As a result, more flexibility for data processing is added to the application with a generic and robust execution and management environment for the data processing services. This is not limited to the services provided by FlexMash but is extended to third party services. This elaboration explains the approach taken for this thesis in order to reach the desired result.

Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

1 Introduction and motivation

In this chapter, the Master thesis tasks, the motivation behind it, and what the end goal was are described.

Introduction

In the current age we live in, IT systems are getting more complex and bigger in size everyday [GS93]. The areas of data processing and integration are no exception to the evolution IT systems are witnessing. As a result of that evolution and growth, the costs that enterprises need to pay to handle their data and be able to properly use it increase [GS93].

Extract-Transform-Load (ETL) processes are one of the most common ways of dealing with data processing and extraction challenges. Unfortunately, traditional ETL processes come with their own shortcomings, such as:

- The level of complexity and effort required to create and use them
- The technical knowledge level required for personnel expected to manage them properly
- Their lack of flexibility needed for modern use cases due to the fact that most of the time, they are executed in a static execution environment [HB16; HM16; HRWM15].

Enterprises already have and need to continually add a variety of data sources, with various structures and semantics, which make it a challenge to handle all of the data in a consistent manner. Building a system that satisfies the needs of the users, while at the same time provides the flexibility needed to handle data and get the desired outcome, is not an easy task. However, the more flexibility the solution offers, the more complex it is to build and maintain.

ETL processes have a statically defined data integration scenario which makes them easier to implement. In order to be able to create more flexible scenarios, data flows could be used along with various other technologies, such as workflow engines, data

integration solutions etc. This would allow for more flexible scenarios based on current needs without requiring complex processes or a deep technical background.

The approach of this thesis takes advantage of data mashups in order to achieve the goals intended. Mashups are defined as pipelines that process and integrate data based on different interconnected operators, that realize data operations such as filter, join, extraction, alteration, or integration [YBCD08]. The main aim is to integrate data, which we obtained from various sources, through the execution of data mashups. The current state-of-the-art creation requires technical knowledge and sometimes programming knowledge to design and execute those operations, which is a disadvantage for stakeholders with no IT background.

To solve this challenge, the extended data mashup approach FlexMash [HB16; HM16; HRWM15], developed at the University of Stuttgart, introduces the concept of patterns to relieve non-IT experts from the technical challenges that arise from needing to implement their own data processing and integration scenarios. This enables them to perform data operations using means specific to their domain. The data processing in FlexMash is done in a distributed manner based on services. These services provide functionality to filter, aggregate, or analyze data and are usually hosted on powerful cloud environments.

A key feature that contributes to the method's flexibility is the ability to add a new data source and data operations to an existing scenario, while at the same time, keeping the integrity of the model by being able to easily regenerate and reexecute it. Current existing solutions provide answers for specific use case scenarios and do not offer a generic solution.

Another challenge that arises is being able to deal with unstructured data. It is uncommon to find a system that can process and integrate both structured and unstructured data simultaneously. The integration of unstructured data can enhance the quality of the information obtained from certain scenarios, but the complexity of that integration comes at a cost.

In this thesis elaboration, the FlexMash approach introduced in [HB16; HM16] is briefly discussed and its uses are expanded upon. New enhancements that build upon the original Flexmash application aim to increase flexibility and robustness for data processing and integration scenarios. The main goals for this thesis are to:

- increase the flexibility of the application FlexMash
- widen the borders of the possible data processing scenarios that the application can handle
- maintain the simplicity of the application, so that users with limited or no technical knowledge are able to use it.

A key component that is essential for these goals is the Service Platform, which is introduced in this thesis elaboration as part of the architecture. The Service Platform is responsible for keeping track of the services available, making sure they are running properly through the health checks, and managing the queries made by the users or the application. It is also responsible for the administration of the services, in addition to keeping track of the services providers and their services. Thus, the Service Platform is used to flexibly add services to the application, which can then be be used in different scenarios. This increases the range of scenarios and use cases of the application, especially while keeping in mind that the services in the application may have different providers.

The remainder of this thesis elaboration is structured as follows:

Section 2 provices the background and necessary information needed to be able to understand the concepts of this thesis.

Section 3 discusses related work and different research connected to this thesis.

Section 4 describes the conceptual approach to the architecture of this Masters thesis, the different components of the project, and how it all comes together to achieve the desired outcome.

Section 5 describes the implementation of the services, includes code snippets, a discussion of the technologies used, and the technical level details of the different components of the project.

Section 6 evaluates how this approach achieves the goals described in the introduction.

Section 7 contains a summary of the approach introduced and possible future work to build on the work done in this thesis.

2 Background

In this chapter, the basic concepts that are needed throughout this thesis are described. The background concepts serve as a foundation for the understanding of the thesis, and are important to describe in detail, in order to get a proper overview of the work done, and the reason behind the choice of concepts and technologies used.

2.1 Pipes and Filters Architectural Pattern

One of the most important elements of building a software system is the architectural design level of that system, as it can enhance or decrease the scalability and portability of said system. Thus, IT professionals have begun to pay more attention and dedicate more resources to to architectural patterns. The architecture for a system separates computation from control on the system's level [Dob03].

System architecture is thus concerned with all aspects of the system's performance, such as: the functionality of every component in the system, the protocols that are used for communication between different components, the processing rates of end-to-end transactions, and hence the overall performance of the system.

In practice, the architectural design of a system provides an abstract level that designers of the software system can use to understand the system behavior without the need to go into details of the implementation. There are some common architectural styles such as: Pipes and Filters, Event-based, Layered Systems, Table Driven Interpreters, etc. [GS93].

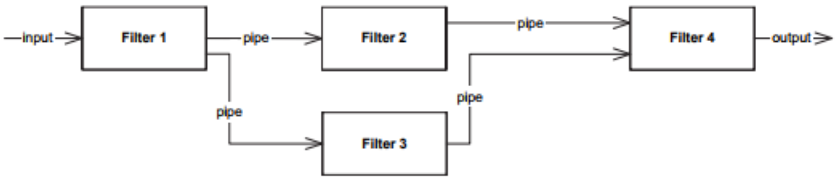


Figure 2.1: example of a simple system in the Pipes and Filters style [AZ05]

In many applications, the functionality is achieved by processing or transforming data, which is done by organizing the steps of the transformation into stages that can be processed simultaneously or in sequence. In this way, the system performs its desired goal more efficiently. This organization in stages has been transformed into a pattern [FO09]. One of the patterns used and built upon in this thesis is the Pipes and Filters pattern [KAB+04], depicted in Fig. 2.1.

In the Pipes and Filters pattern, all components are either filters or pipes. The filters transform the data provided as input and have typed input and output interfaces. The pipes act as connectors between two filters, through which data is transferred using a chosen data transfer protocol. Each pipe has two interfaces: the input interface and the output interface in order to be able to send and receive data [MKMG97].

The benefits gained from using the pipes and filters architectural patterns is visible in scenarios, where large amounts of data need to be processed quickly. Examples include web servers, message processing applications of large enterprises, and business process engines [GK04]. In our case, there are numerous processing steps that need to be performed for data processing and transformation in user-defined sequences of processing, hence, the applicability of the pipes and filters pattern.

Simply put, the filters can have inputs from other filters (or another data source) and the output is sent to another filter through the pipes. Hence, each filter is a processing unit along the chain that performs a certain step. This pattern uses chains of filters as described in Fig. 2.1 to exploit the key aspects of the design mentioned in [Dob03; GK04; MKMG97]. These include:

- Uniform communication means between the filters
- The statelessness of filters to avoid the overhead resulting from maintaining the state of the data
- Ease of reuse and recombination of the component which adds flexibility
- Elimination of the need to keep intermediate results in files

There are different variants and extensions of this pattern, which entails some decision making that would enhance the overall performance of the system. Some examples of those decisions would be whether to use Push or Pull protocols for the pipes and filters, a combination of both, or the use of filters with more than one input/output, etc.

Three example refinement strategies mentioned in [PR99] are blackbox or behavioral refinement, structural refinement (glassbox refinement), and signature refinement. The black box refinement is related to systems where the components are black boxes with no details about how they function. The structural refinement can be addressed by providing a subsystem architecture.

The manipulation of the components' interfaces is dealt with through signature refinement. In our case, the use of the pattern is done by giving the engine a model that is a directed acyclic graph. It consists of adhoc services used to manipulate and format the data, while the pipes in the scenario represent the transmission of data between the different filters through service discovery. This is discussed in more detail in Section 5 which covers the implementation details. Some examples of known uses of this pattern were mentioned in [FO09] and include: Microsoft's BizTalk Server 2004, and the Apache Cocoon web development framework, etc.

2.2 Service Oriented Architecture

Companies nowadays are facing the challenge of ever increasing IT costs, and they must find ways to decrease said IT costs, while at the same time maintaining good customer service [Ran03]. This is further compounded by the following challenges:

- heterogeneity which is a result of the widespectrum of different systems
- architectural patterns of legacy systems and applications, which pose a challenge especially when the enterprise tries to integrate products from different vendors.
- changing customer needs and requirements which leads to the shortening of the product cycles in order to be able to improve the time to market.

IT experts are thus always trying to develop more flexible and responsive platforms in order to meet these challenges, by trying to build applications and platforms that are: loosely coupled, location transparent, and protocol independent. That is where Service Oriented Architecture (SOA) comes in, because the consumer of a service does not need to care about the infrastructure or technicalities of the service.

Particularly, technical specifications from different implementation technologies such as J2EE or .NET should not affect SOA users [EAA+04; PH07]. A component in service oriented design is defined in [EAA+04] as "*an executable unit of code that provides physical blackbox encapsulation of related services. Its service can only be accessed through a consistent, published interface that includes an interaction standard. A component must be capable of being connected to other components (through a communications interface) to a larger group*".

By treating components and their services as a black box, we don't need to worry about different implementations and technical details, since the services could be interacted with through a loosely-coupled and message-based communication model.

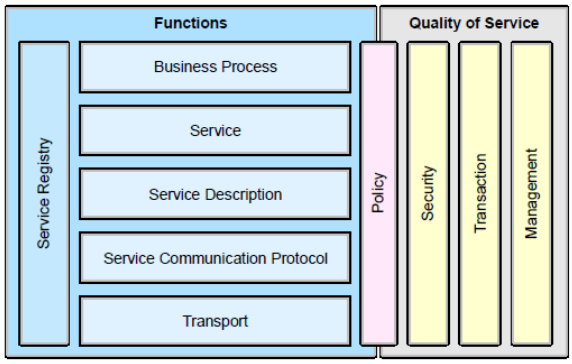


Figure 2.2: SOA elements [EAA+04]

SOA takes care of both functional aspects and the quality of service actions as described in Fig. 2.2. The focus will be on the functional aspects described in the figure as they are a critical part in the development of this thesis. Briefly, the functional aspects include:

Service Communication Protocol which is the mechanism agreed upon to exchange requests and responses between the consumer and the provider of a service.

Service Description that is an agreed upon schema for the details of the service such as its inputs, how it could be invoked, and what it does for a successful service transaction.

Service Registry that acts as a repository for the services and their descriptions, is used by providers to advertise their available services, and for consumers to discover available services.

SOA is focused on solving problems such as application integration, transaction management, and security policies. The driving goal of SOA is to ease application integration through solving or decreasing the impact of the challenges. This enables and supports the flexibility and agility that IT experts try to achieve while building applications and systems.

Through SOA, service consumers, regardless of what operating system or programming language they use, can access a SOA service to create a new business process. Logically, a service in a SOA system consists of two main parts: service interface and a service implementation [PH07]. Because interfaces are platform independent, a client from any communication device, using any computational platform, operating system, and any programming language can use the service.

How the different functional components interact is described in Fig. 2.4. In order for a consumer to interact with a service, the consumer needs to first find the service. This is

done by communicating and querying the service registry, which provides the details of the service to the consumer assuming, of course, that the desired service exists.

The service details include the description and the address of the service, which the consumer then uses to be able to communicate with the actual service.To be able to achieve the interaction described, first, services need to be published so that the consumers are able to query them and then bind and invoke them using the information provided. By using this architecture, services provide many advantages, which include that they are self-contained, support interoperability, loose coupling, and have an interface that is used for consistent communication.

SOA became one of the popular solutions to the challenges facing enterprises due to the fact that it provides a level of abstraction that allows companies to reuse existing assets instead of building components from the ground up. It is also easier for integrating services to form new scenarios, since it minimizes the impact of changing implementations of the infrastructure. As a result, SOA helps in reducing the costs of the IT enterprises while at the same time providing a means to be more responsive and enhance the time to market.

2.3 Universal Description, Discovery & Integration (UDDI)

The Universal Description, Discovery and Integration (UDDI) is defined in [SRAW03] as "*specification for distributed Web-based information registries for Web Services*". Through the (UDDI) specifications, a platform-independent way of describing and discovering Web services and their providers is made available to consumers [CJCR04].

UDDI consists of three components: "white pages" which contains the basic contact information and identifiers for a company, "yellow pages" to allow companies to be listed according to their industry categories, and "green pages" to hold and maintain interface details of how a Web service can be invoked [SRAW03].

The UDDI data model is centered around XML with the data types, and different types of registries (depicted in 2.3) to represent businesses, their available services, and technical descriptions of the services [Dra01].

As for the data manipulation, it is completely based on XML and uses SOAP as a protocol. UDDI could be implemented as a group of distributed registries that are kept synchronized. The role of the UDDI is a key element in providing the loose coupling needed. This is why many approaches, such as the ones mentioned in [Dra01; SPAS03; SRAW03; ZLCC02], are trying to improve the performance and ease of access to such a component.

The approaches vary from automatic service discovery (like the approach in [SPAS03]) to enhancing the performance and managing to get better flexibility and efficiency (like the approaches mentioned in [CJCR04; Dra01; SRAW03]). In the end, the importance of the concept itself is to have a centralized component that takes care of services listing and discovery.

Type	Description
Public	Querying and matching information is public to all web service consumers. In that sense public UDDI appears to be a WS itself. Publishing information into the registry is supported through secure channels (e.g. https), but this does not spoil its public character. Data communication with other registries is supported.
Protected	The notion of trust between collaborators characterizes this kind of registries. Such registries are implemented within a closed-group environment with monitored access to third parties. Administrative features may be delegated to trusted parties. Data communication with other registries is allowed only if explicitly specified.
Private	It is about fully secured isolated registries. They are usually domain specific registries in an internal network. Data communication with other registries is not allowed.

Figure 2.3: UDDI Registry Instances [GPST06]

2.3.1 Service Discovery

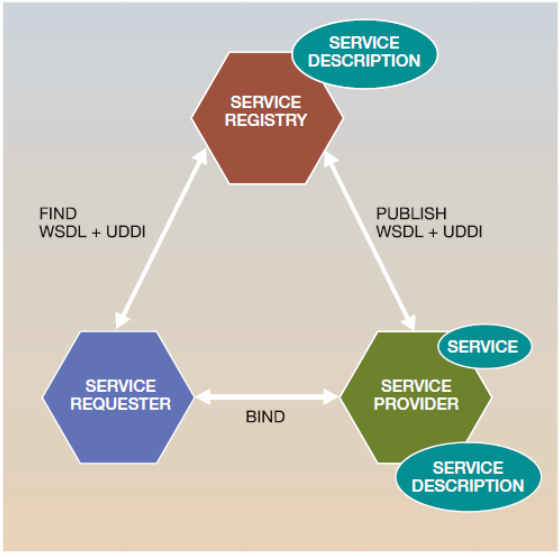


Figure 2.4: SOA service discovery [GS93]

Service discovery mechanisms are defined in [SKWL99] as the matchmaking process of services. The challenge is to find the best match for a service required by a consumer from the available services offered from different providers. This is done through a middle agent [DSW97]. It is a challenge to find the best available service that matches the requirements needed for an application, since it will affect the overall performance of the system as well as the end result.

Unfortunately, the approach based on UDDI requires time and patience from both providers and the consumers of the services in question [GPST06]. Keeping in mind that a great amount of data online is available via APIs, which expose the data as JSON or XML, combining code that matches different data into one application is not the best practical way to build up systems that depend on multiple data sources. This why service discovery and uniform access to multiple data sources is important.

Most online published data uses web standards such as URIs, HTTP, XML, SOAP, REST, etc. [HNP+11]. It all stems from SOA, with the end goal being more flexible and loosely coupled systems. Thus, efforts are directed towards different aspects of the service discovery problem. In [MK10], an enhancement over an existing three-tier [PH07] SOA architecture is proposed, that reduces the number of exchanged messages, by adding a new layer for the web requests' preprocessing. Providing uniform access to the services is also important to decrease the interface's development effort, which is the approach suggested in [FHA+99; GPST04].

The biggest challenge affecting service discovery mechanisms is the heterogeneity between the offered services [GPST06]. It can be seen on many levels including:

- Technological heterogeneity which entails different platforms and/or different data formats.
- Ontological heterogeneity that could be seen in domain-specific terms within services that can differ from one another. This is most obvious when comparing services from different vendors.
- Pragmatic heterogeneity that results in different implementation of domain-specific processes and their support.

A lot of efforts are directed to having a uniform, automated access to the best services available, some of which are proposed in [AM07; HNP+11]. In this thesis, the effort for service discovery is developed based on the concepts of having a central component that holds all the information of the services the platform offers as shown in Fig. 2.4, while having the application (Flexmash) connecting to it, in an automated way based on the workflow provided by the user.

The platform also allows access to the services from outside the application, thus, serving more consumers while maintaining the status of each service and whether it is reachable or not, providing only functional services.

2.4 Enterprise service bus (ESB)

With the rise of SOA, a new need appeared for an infrastructure that is able to combine Message-Oriented Middleware (MOM), web services, transformation, and routing intelligence, in order to be able to support and enhance SOA. There are multiple definitions for an ESB but the main concept remains the same. A software service can change over time due to the increasing growth in the IT environment, their signatures or data types might change among other aspects that are susceptible to change.

In a large IT-system, it is extremely difficult to have a system built with the same technology, logic, protocols, and components that can handle all aspects of the functionalities required by it. It is even harder to maintain such a system if it even exists. Because such a system existence is impractical and highly improbable, it has to consist of different components which by default entails different protocols and technologies. Assuming that different components of the system communicate with point-to-point integration, it would be an extremely complex system and the costs for maintenance of such a system would be extremely high. Essentially, there are two options for building the communication infrastructure for large systems:

1. Building and designing each component to be able to communicate and integrate with all other components within the domains of the company
2. Building a communication layer to manage the logic and integration between the different components (the ESB)

Assuming the decision was made to work with the first option, this would mean that each component needs to have an interface for each other component. As a result the system would be tightly coupled. In case a new component is introduced to the system, the overhead for creating new interfaces for all the components (to be able to communicate with the new component) would be cumbersome. This makes the second option the more logical one for building a system, while maintaining the costs of development and maintenance much lower than the first option.

In order to have a functional ESB, it needs to implement certain functionalities. Some of those functionalities are mentioned in [Lie13; Men07; PH07], and include:

Message Transformation For a large system, it is common to have different components with different messaging formats, so the ability to transform the messages from the source format into a format that the target would understand is a key feature to avoid developing N*(N-1) interfaces for the components assuming that we have N components.

Intelligent Routing The task is to deliver the messages sent from the consumer to the correct intended service providers, and then send the responses back to the right consumer. There are several examples of routing, such as content-based routing or routing specific messages to several destinations where the consumers subscribe to a certain topic.

Adapters which are used to connect to the native APIs and data structures that the applications expose. Many ESB solutions offer a wide range of adapters.

Process Orchestration may be included in an ESB, which is an engine that is able to execute business processes described with the Web Services Business Process Execution Language (WS-BPEL)

Reliable Messaging to be able to queue messages and ensure a guaranteed delivery to the destination, while being able to respond back to the requester if needed. That includes synchronous and asynchronous messages.

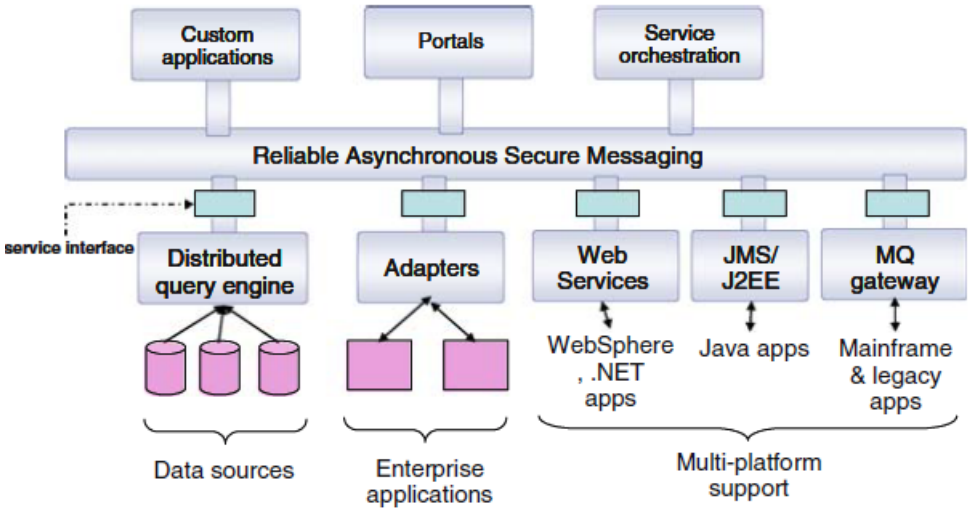


Figure 2.5: Enterprise service bus connecting diverse applications and technologies [PH07]

Different approaches were made to extend an ESB, such as in [Muh12], for Multi-Tenancy support. Regarding the evolution of the ESB in general the discussion in [Lie13], emphasizes the importance of the concept. In Fig. 2.5, an example architecture of an

ESB is shown, where the ESB integrates a .NET application using a C# client, J2EE application using JMS, an MQ application that interfaces with legacy applications, in addition to external applications and data sources using Web services.

As the figure shows, the ESB allows a more efficient integration of components, where they only need to have interfaces to communicate with the ESB, not interfaces for every other component. It also provides abstract connection information for physical destinations which allows communications to use logical names instead of IPs and ports. ESBs can also be implemented in different protocols such as HTTP, JMS, etc. [EAA+04]. It is up to the system architect to decide the best way for communication between the system's components, based on the end goal of the system, and the features of the components. Three alternative ways of communicating between an ESB and an application are mentioned in [KAB+04; PH07]:

1. Using applications that provide a Web service interface where the WSDL defines the interface to communicate directly with the application business logic.
2. Using a non-web service that interface where the application does not expose business logic via Web services. An application-specific adapter can be exposed by the application to act as an intermediate layer between the application API and the ESB.
3. Using a service wrapper that acts as an interface to adapters. In some cases the adapter may not supply the correct protocol (JMS, for example) that the ESB expects. In this case, the adapter would be Web service enabled.

2.5 Business Process Modeling Notation (BPMN)

BPMN is a proposed process modeling technique, which was developed based on revisions of other notations such as UMO, IDEF, ebXML [RIRG06]. It was developed as a result of the demand for having a graphical notation for modeling processes. The main goal of BPMN is to make it easy to be clearly understandable by all relevant stakeholders, starting with the business users who create the requirements and first draft of the processes, the technical developers who implement the technology responsible for performing the processes, and finally the business people responsible for monitoring them.

BPMN defines a Business Process Diagram (BPD), which is based on a flow charting technique tailored for creating graphical models of business process operations [Whi04]. The BPD makes it easier for users and stakeholders with no technical background to understand the flow of the processes, without having to understand the low level

implementation details, providing them with the ability to also create understandable diagrams. BPMN also allows models to extend some of its basic notation, which is a feature that enables more flexibility for different businesses to use it.

BPMN could be used to communicate a variety of different information to different users and audiences. This includes various types of modeling, with different levels of details. There are two basics types of models that BPD can be used to implement: Collaborative (Public) B2B Processes and Internal (Private) Business Processes. The B2B processes express the flow of interactions between business entities (two or more), which offers a bird's-eye view over the entire interaction flow. Internal business processes focus more on a narrowed down view of a single business organization, despite the fact the internal processes often interact with external services and participants.

Modeling patterns are also available for users [All16], since similar problems are encountered repeatedly by the people that need to model processes. The patterns offer a way to model those recurring cases, and to save time by not having to develop specific solutions each time. A good feature that could be used with BPMN standards is choreography modeling. A choreography model contains the relevant interactions and dependencies [Rec08], which can be done through the different control patterns described in [WAD+].

2.6 Data Mashups

According to [YBCD08], Web mashups are Web applications developed using contents and services available online, with the goal being to combine sources to create a new application or service. The HousingMaps[1] application is an example of a successful mashup. It serves as an assistant to people who are moving from one city to another and are searching for housing, by combining real-estate data with map data. A similar application could be developed using conventional technologies, but by using mashups, even end users can create useful mashups from data sources without any in depth technical know how.

Several mashup specific development tools and frameworks have recently emerged that increase the speed of the mashup development process, and at the same time give the end users the ability to develop their own mashups. Some of these tools are existing

[1] (www.housingmaps.com)

mashup solutions that were mentioned in [HRWM15; YBCD08] such as: Yahoo Pipes[2], IBM Mashup Center[3], Intel Mashmaker[4] and OpenIoT Mashup[5].

Mashups are generally created to aid with reusability and ease of access, and so end users will be able to compose their own mashups. That being said, in [HRWM15] limitations for the previously mentioned tools were found and tackled in regards with coping with different requirements of various scenarios, usability by non IT-experts, handling of heterogeneous data, and scalability.

2.7 Camunda engine

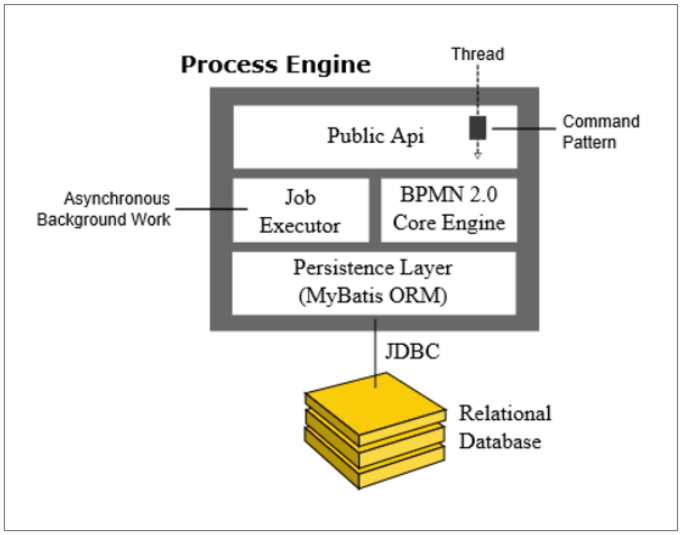


Figure 2.6: Process Engine Architecture[6]

The core of Camunda BPM is a model execution engine that supports the OMG standard BPMN 2.0 for process automation. It supports key concepts such as:

- Flexible implementation through several extension points and customization.

[2]http://pipes.yahoo.com/pipes/
[3]http://pic.dhe.ibm.com/infocenter/mashhelp/v3/
[4]http://intel.ly/1BW2crD
[5]http://openiot.eu/
[6]www.camunda.org

- Job executor which helps in optimizing the performance and scalability.
- Transactions management, i.e, whether it's done through the engine itself or through a platform transaction manager.

Hence, this engine was used for executing the models in this thesis, which will be discussed in details in later sections. It also applies persistence through a number of strategies mentioned in their documentation. The Process Engine Architecture in Fig. 2.6 facilitates its integration with the application FlexMash. Using the public API provided, which is a Service-oriented API, allows Java applications like FlexMash to interact with the process engine. Using the API, the application can control the different responsibilities of the process engine (i.e., Process Repository, Runtime Process Interaction, Task Management, etc.). The core engine then transforms the BPMN XML files generated from the transformation of the Mashup Plans into Java Objects and a set of BPMN Behavior implementations such as Service Tasks and Gateways.

3 Related work

In this section, the basic concepts required for the development of this thesis are described in more detail. The related work serves as foundation for the concepts on which this thesis is built, and it is important to describe in order to get an overview of the work done and the reason behind the choice of concepts and technologies used.

There are many approaches for using Mashups to process data, which are characterized in [YBCD08] by looking at the components and how they are glued together i.e, the composition logic. The component model is what decides the nature of the components, as well as how they can be integrated together. In [YBCD08] components are characterized using three properties: type, interface, and extensibility. The composition model on the other hand, is what determines how the components are integrated. It has many distinct characteristics that could be used to better understand it.

3.1 FlexMash

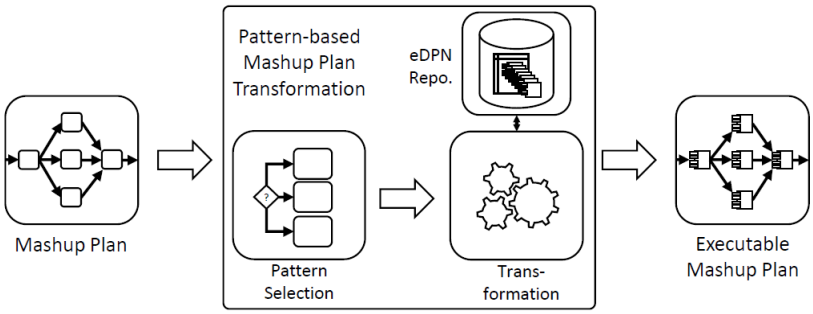
FlexMash is a proposed mashup approach in [HB16; HM16; HRWM15] that consists of five main steps:

1. the modeling of Mashup Plans
2. the selection of transformation patterns
3. the pattern-based transformation of Mashup Plans into an executable format
4. the cloud-based data mashup execution based on user requirements
5. the storage and/or visualization of the derived result [HM16]

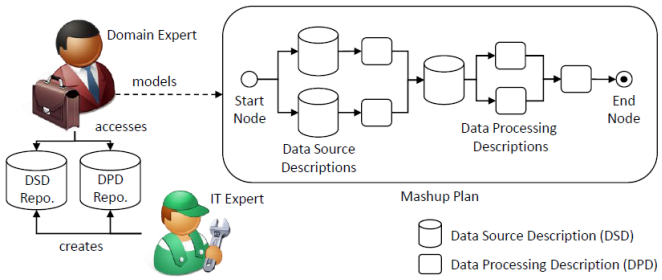
In this thesis, the following steps are discussed (2, 3, and 4), since they are where the contribution of this thesis comes in, for the enhancements of the application FlexMash. It divides the data mashup into four abstraction levels. These are the model, its transformation, execution, and presentation level. It successfully enables the abstraction from the non-technical, domain-specific modeling to the technical execution. Hence, FlexMash can be considered an applicable data mashup solution.

The Pipes and Filters pattern is used to take advantage of the software components it creates, and enable easy interconnection between them for different possible scenarios. This means, that after the graphical model is created, it is then transformed into an executable format such as, e.g., a workflow model. Using the executable model, the functionality of the filters is provided by services, that are called in the order as defined by the data mashup model [HB16].

3.1.1 Mashup plans



(a) Mashup Plan Transformation Components [HRWM15]



(b) Overview of Mashup Plans [HM16]

Figure 3.1: Mashup plans

Mashup Plans (shown in Fig. 3.1b) that are related to the Pipes and Filters are introduced in [HRWM15], and defined as "*non-executable, domain-specific model to define data mashups* [HM16]". This is essentially a directed, cohesive flow graph containing nodes and edges, where a node is either a Data Source Description (DSD) or a Data Processing Description (DSP).

The mashup plans need to have at least one DSD and one DSP, which is one of the restrictions for a model to qualify as a mashup plan [HRWM15]. Throughout the development of this thesis, the approach for mashup transformation and execution is built upon as explained in 3.1a. Since the modeling of a mashup plan abstracts the

technical details of the plan and how it can be executed, a technical model needs to be generated that translates the plan into an executable model. Different transformation patterns are also introduced in [HRWM15]. For this thesis, the Robust Mashup pattern was used.

3.2 Online Citation Service

In [TAR07], an approach for a complex dynamic data integration mashup framework is introduced. Using Mashups, this framework offers data transformation, query generation, and online entity matching capabilities. The scenario used to illustrate the approach is an Online Citation Service[1] (OCS). The example is an implementation that combines bibliographic data to dynamically calculate citation counts for venues and authors. OCS allows the generation of citation counts for publication lists of authors and venues. It obtains the publication lists from the DBLP bibliography[2] and the citation counts are obtained from Google Scholar. The citation counts from Google Scholar (GS) are obtained dynamically based on the scenario at hand.

The framework enables developers to create their mashup algorithms as scripts, where functionality of a mashup script is also available as a web service. This is so that it can be used by a web interface or by another web service, e.g., within another mashup.

Mashups are developed using a high-level script language, which also uses XML data structures. The framework also offers several operators for data transformation, e.g., fuse, aggregate, and set operations e.g., union, intersection, and difference. Fuse transformation functionality for example, performs object matching and merging. Fuse takes as input two XML documents A and B representing various objects, and identifies objects referring to the same real world entity.

3.3 Mashup tools analysis

[DHPB09] presents an analysis of the richness and weaknesses of various Mashup tools with respect to their data integration aspects. The analysis included seven Mashup tools, chosen based on the tools' popularity when the analysis was performed. In addition, the tools' availability was taken into account to ensure that sufficient experimentation and accurate reporting results could be gained for the study. The objective was to offer a

[1] http://labs.dbs.uni-leipzig.de/ocs
[2] http://www.informatik.uni-trier.de/~ley/db/

view on the state of those tools, and understand the general approach to developing them. The tools considered in the study were:

1. Damia, a Mashup tool provided by IBM. This tool focuses on data feed aggregation and transformation in enterprise environments.
2. Yahoo pipes, a web-based tool provided by Yahoo. Users can build mashup applications by aggregating and manipulating data from web feeds, web pages, and other services.
3. Popfly, a web-based Mashup application by Microsoft. It allows the users to create a Mashup combining data and media sources.
4. Google Mashup Editor(GME), a Mashup development, deployment and distribution environment by Google. The Mashup can be created using technologies like HTML, Java Script, CSS along with the GME.
5. Exhibit, a framework for creating web pages containing dynamic and rich visualizations of structured data.
6. Apatar, a Mashup data integration tool that helps users integrate desktop data with the web.
7. MashMaker, a web-based tool by Intel for editing, querying and manipulating web data. It allows users to create a mashup by browsing and combining different web pages.

According to the study, the majority of the tools have an XML-based internal data model. This is due to the fact that most web-data is mainly exposed in XML format [DHPB09]. The other dominant internal data model present in Mashup tools is object based. In order to manage data, the tools offer a small set of operators for integration and manipulation of data. Those operators are offered based on the main goal of the tool.

It is worth mentioning that none of the analyzed tools implemented a Push strategy for data refreshing. The reason for this is that the majority of the currently available APIs are REST based, which in turn entails that the communication is initiated by the client and not the server. This makes maintaining the state of the connection in this case not achievable. Instead, the tools used a Pull strategy to maintain the integrity of the data.

According to above mentioned study, the majority of tools do not support the reuse of created Mashups. Instead, they focus on offering services to process and manage data. Finally, all the tools are meant to target “non-expert” users, but some programming knowledge is usually required. This limits the pool of users capable of using the tools to those with some basic to advanced programming knowledge, that are able to develop the mashups using programming languages such as Java Script.

3.4 SOA-based Mashups

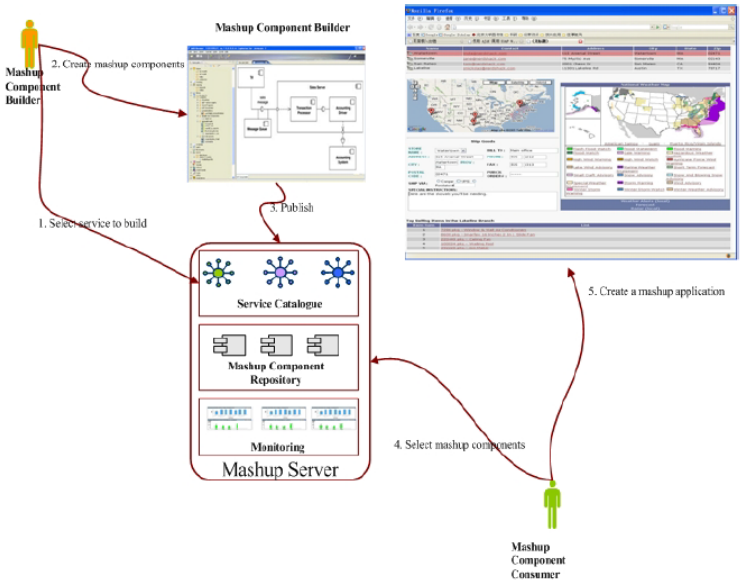


Figure 3.2: SOA Extension with Mashup [LHSL07]

A Mashup application consists architecturally of three different participants: API/content providers, the mashup host, and the consumer. It is a process that integrates data from different sources, and therefore, it is a service composition style from a SOA perspective. In [LHSL07], an approach is introduced to extend the current SOA model with Mashups. In addition, a new mashup component model is proposed. The approach extends the basic SOA roles (provider, broker and consumer), according to the Mashup architecture and roles. Mashups are thus in this case an extension of the SOA paradigm, which entails a similar life-cycle to that depicted in Fig.3.2.

For Mashups, this approach extends the service composition style of SOA to the interface level, which is at the application level in traditional service composition cases. This means that the users have access to simpler techniques for composing a scenario, which decreases the need to have a technical background.

The component model introduced is classified into three elements:

1. **UI component:** the part the user interacts with, usually consists of the UI components that hide the service components' details. This abstracts the details of the composition, hides it from the users, and enables them to only deal with the UI.

2. **Service component:** deals with data manipulation and processing. It can vary in the implementation details for the same logic.
3. **Action component:** acts as the connector between UI and Service components, as it defines actions driven by events like a button click on the UI for example.

Since users can focus on the UI level composition, the data exchange between two mashup components is processed by their own Action Components. This leaves the detailed generation of the code in the hands of the browser. Once a component is chosen and used in the Mashup, the code corresponding to the component is added in the background. The total model is then used to execute the scenario based on the code generated. This provides users with the ability to experiment safely with powerful tools, without affecting the usual IT process.

4 Conceptual approach

In this section, the architecture of the masters thesis, the different components that were developed, and how they work together are described. The architecture of the application, and how the different components fit together are also further expanded upon.

Through the development of this thesis, a comprehensive set of data services is created. This set contains basic functionality to process data, covering the most important scenarios. It is important to highlight that the development of these data services is done in a generic manner to ensure their re-usability. These data services consist of functionalities to filter, aggregate, and analyze data. Furthermore, a concept for uniform interfaces and a uniform data exchange format of the data services is proposed.

4.1 Overview of the approach

In order to achieve the goals mentioned before, the FlexMash application needed to be extended to make it compatible with the desired Service Platform. Furthermore, the processing of mashups needed to be adjusted, in order to be able to make it more generic, and for easier integration with the services provided.

Removing the need to adjust the application itself with the addition of a desired new service, provides more flexibility to the overall flow of the application. The work done in this thesis consists of:

- Creating a Service Platform that acts as a centralized repository for the services available for FlexMash.
- Extending FlexMash in order to use Camunda BPMN engine in a generic way for the services provided through the Service Platform.
- Integrating the Service Platform directly into FlexMash to be able to use the services as well as making it reachable from outside the application.
- Creating and integrating prototypical implementations of data services to provide the functionalities of data Filtration, Aggregation, and Classification.

4.2 BPMN Model Transformation

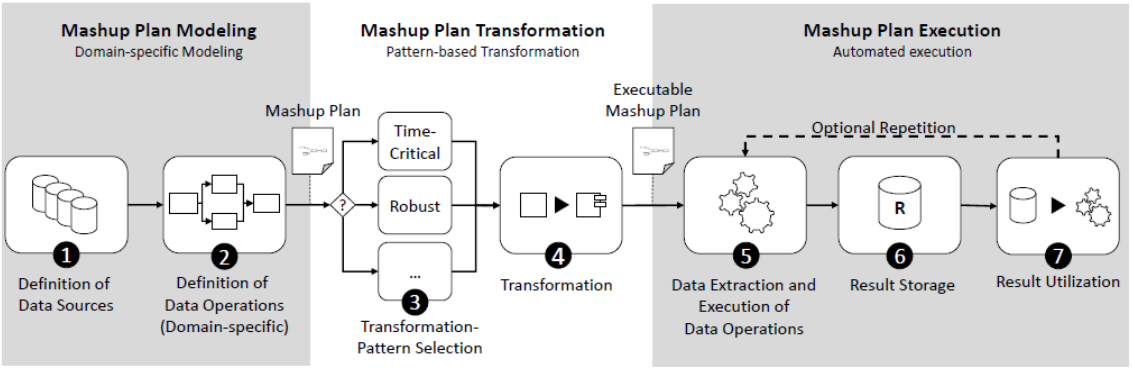


Figure 4.1: Extended Mashup Approach [HRWM15]

The approach for the extended mashup on which the application FlexMash is built is explained in Fig. 4.1. The main contributions of this thesis for the FlexMash application (in addition to the Service Platform) are done for the method's steps: Transformation, Data Extraction and Execution of Data Operations, Result Storage, and Result Utilization. The plan transformation is done via a JSON format the front-end sends to a BPMN workflow, which the Camunda BPMN engine can execute. The JSON format can be found in Listing A.1.

When the user starts executing the designed model, the data is sent from the front-end application engine through HTTP. The JSON data sent is a representation of the model, with the user inputs, connections between the different nodes, and front-end styling elements saved as a template. The back-end extracts the relevant data from the file in order to be able to transform it into an executable BPMN model. For a successful transformation, the nodes of the model, node inputs (if they exist), and the flow or connections between the nodes are needed. The styling data is ignored since it is not needed for the actual execution.

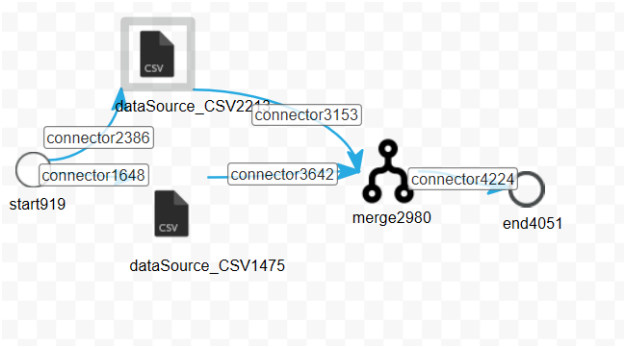
Using this information, the application transforms the model sent by the front-end, into a BPMN model that can be executed using the Camunda BPMN engine. The reconstruction is done by traversing the JSON data, and building the BPMN model, node by node, until all the nodes are represented and connected in the same way. This process results in a logically equivalent model in BPMN notation. An example of that transformation process is depicted in Fig. 4.2, with a sample scenario (Fig. 4.2a)and the logically equivalent BPMN model (Fig. 4.2b).

When comparing the two models depicted in Fig. 4.2, it is obvious that in Fig. 4.2b two extra components – logical gates – are present, unlike in Fig. 4.2a. The logical gates are a necessary adjustment made during the transformation to obtain a sound execution of the model. For parallel paths, as shown in the figure, the BPMN engine requires the presence of the parallel gates in the model to properly work.

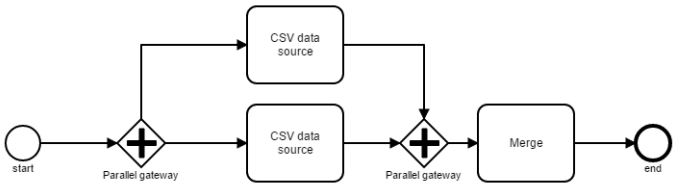
The functionality of the gates is to act as a semaphore that waits for both branches to finish execution before starting the final execution of the node at the end of the forked path. During the transformation, the nodes with multiple incoming or outgoing transitions must be kept track of to ensure that the parallel gates are correctly placed in the right position in the model.

The idea behind the replacement is to have the parallel gates directly after the nodes with multiple outgoing transitions, and again directly before the final nodes with multiple incoming transitions. The parallel gateways are what synchronizes the execution of the multiple branches, and doesn't execute the final merge until all of them are finished executing. Without the parallel gateways the merge functionality at the end would be executed multiple times, once for each brach.

When it comes to the user, the nodes with multiple transitions and the parallel gates are treated as one unit, which is handled by the transformation algorithm, so that the user never needs to model extra nodes or take care of the multiple branching. This allows the process to be simple for the end user.



(a) FlexMash model



(b) Result BPMN model

Figure 4.2: Transformation example

The workflow generated is then processed and executed based on the various data sources and services used. The automated execution is done through a generic execution that communicates with a service registry, gets the information of the service that should be used, and communicates with it. The results are then returned to the engine for further processing and to continue with the flow of the generated model.

The transformation process takes care of the multiple branching in a workflow, in order to take the burden off of the user when modeling a scenario. Since the BPMN engine requires adding special gates before and after each multi branch path, the transformation phase models the Mashup Plan with the appropriate notation.

4.3 Service Platform

IT giants such as Google, Amazon, Microsoft, IBM have started providing various services online, thus saving a lot of users the costs of doing the computations they need on their own. It saves the users from the hassle of self software deployment and maintenance. Providing Software as a Service (SaaS) [TSB10] is a rapidly growing market due to

a variety of reasons, one of main ones being that, with SaaS, users do not have to incorporate the entire implementation and service costs of those services into their budgets.

With that growth of online services, it is now easier to integrate different sources of data and functionalities into mashups [MRG08]. Depending on their level of maturity, the platform contributes to decreasing the development effort and reducing operating costs [CLPZ]. The Service Platform developed for this thesis consists of three main parts: the services management infrastructure, service registry, and the actual services.

4.3.1 Infrastructure

For the infrastructure, upon which the Service Platform was built, I used Consul[1]. It is a Service Discovery and Configuration platform that provides a lot of useful features that I used for the Service Platform. It has multiple components, but as a whole, it is a tool for discovering and configuring services. It provides several key features:

- Service Discovery: Services could be provided to Consul, and Consul enables the consumers to discover providers of a given service. Using either DNS or HTTP, applications can easily find the services they depend upon.
- Health Checking: Service providers could also supply Consul with health checks associated with a given service ("is the webserver returning 200 OK"). It is used by the service discovery components to route traffic away from unhealthy hosts.
- Key-Value Store: Applications can make use of Consul's hierarchical key/value store for any number of purposes, including dynamic configuration, feature flagging, coordination, leader election, and more. The simple HTTP API makes it easy to use.
- Multi Data center: Consul supports multiple data centers out of the box, which eliminates the need for building additional layers of abstraction to grow to multiple regions.

Consul is a distributed system, which enables the Service Platform to grow with no extra effort of development. Consul nodes running on different instances run an agent, which is responsible for the health checks of the services on the node. While at the same time not affecting the discovering of services since it is not needed for that. The function of the agents is to talk to the available Consul servers, where all the services-related information is stored and replicated. It is the job of the nodes to elect a leader, while at

[1] www.consul.io

the same time providing the possibility to run it only on one node. For queries regarding a service, the agents or the nodes could be used where the agents forward the requests to the servers. In case of a cluster, the service discovery requests are sent by the local server to the remote data centers and returns the results. This provides consumers with a level of abstraction of the infrastructure and the flexibility of configuration, since they do not need to contact each server separately.

The features it provides add value to the robustness of the Service Platform in different ways. Service discovery and configuration is one of the main challenges that faces SOA compliant solutions. In many cases, organizations build their own solutions, which adds a lot of costs and overhead to develop and maintain. Hence, the use of Consul for this thesis in order to eliminate those costs. By exploiting this open source technology, more efforts are directed towards the actual goals of the system instead of reinventing the wheel.

Despite the fact that it provides a lot of useful functionalities, I needed to build a layer of communication that is responsible for choreography of the desired interactions. This layer is needed to be able to maintain a consistent choreography, without giving consumers the ability to interact directly with Consul itself. Some of the constraints that this layer provides are ensuring the users to have access tokens to maintain each user's services, and to ensure that the administrative actions done on services are being executed from the owner of that service. It could also be used in future work to keep logs and relevant information for analysis and reference to be able to extend the platform further.

4.3.2 Service registry

The service registry is the layer which the service providers and consumers interact with. It uses the choreography depicted Fig. 2.4. It provides service consumers with the available services, and their details that are given by the providers. It also makes sure that the returned services are healthy and functional by using the health checks that the infrastructure provides. Thus, making sure that the results returned to the consumers contain reachable services only.

As for the service providers, it is responsible for keeping track of the services provided and the operations executed. By providing service providers with tokens, each token holder can only control the services registered to this token. The data is stored in Consul but the authentication and choreography is taken care of by the registry. It is worth mentioning that both consumers and FlexMash application are able to communicate with it, which expands the services paradigm that could be provided for both. The authentication logic and the service queries are taken care of by the Service Platform,

while the Consul platform is used to maintain the data, and perform the health checks for the services. Using the interfaces provided by Consul, the platform orchestrates the flow of the functionalities it offers.

4.3.3 Services

For this thesis, I implemented three services for data processing that are hosted on the Service Platform itself. These services are mainly developed as proof of concept, and the scenario I used them for is to process data in the CSV format. The services as mentioned before are filtration, aggregation, and data analysis. These services are contained and hosted in the Service Platform.

In order for them to be used through FlexMash, they have to be added as nodes to the application. Adding them as nodes also entails registering/publishing them on the service registry through the application. Since users can add nodes directly through the application, all the registered services from different instances, will be available through the service registry, making it possible for other users to incorporate these services, if needed, into their own Mashup Plans.

The filtration service supports simple AND/OR operations that filter the columns in the CSV data and returns the result. An example of the data I used is depicted in Table A.1. The conditions applied to for filteration can be AND conditions or OR conditions, but the service I implemented does not support a mix of both operations. An example condition can be (sex = F AND reason = course) which if applied to the data in Table A.1, it will filter out all rows except those which fit the condition.

As for the aggregation service, it creates an SQL DB instance in memory with both CSV data sets to be merged. After that, they are transformed into relational tables. It also supports simple AND/OR operations. Instead of implementing the merging functionality from scratch, I took advantage of the capabilities of SQL DBMS since it would implement the merging logic efficiently to return the appropriate result.

Then, the SQL instance is deleted from memory and the result then is transformed back into CSV that could be used in other services. For the classification services, I used the Weka[2] library for data mining to implement the Apriori algorithm. The Apriori algorithm generates association rules on the CSV data set that is used as an input for the service. In the implementation section, I will use an example to demonstrate the entire flow, including the services that I implemented for the data processing.

[2]http://www.cs.waikato.ac.nz/ml/weka/

4.4 Components integration

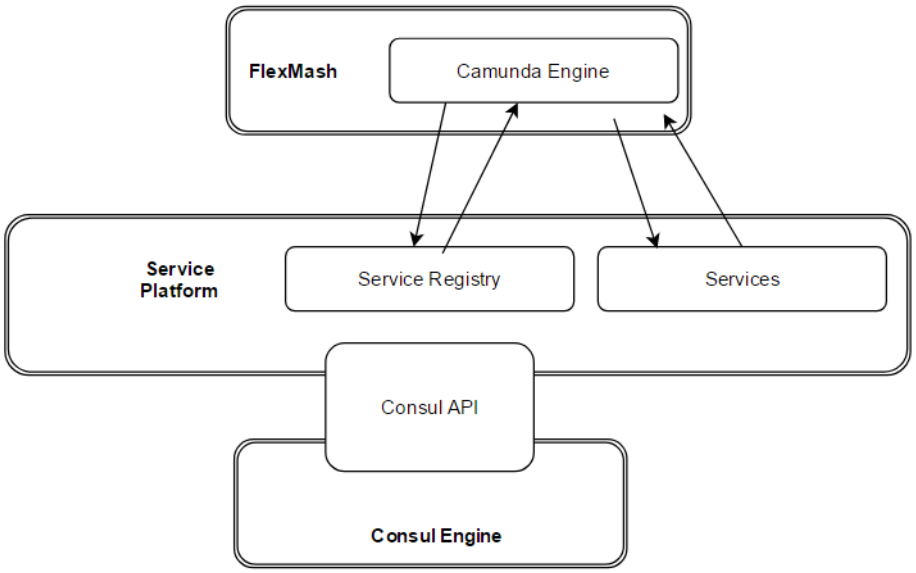


Figure 4.3: Thesis contribution

One of the main goals of this thesis, is to provide a generic way to extend the scenarios that FlexMash is capable of executing. In order for that goal to be achieved, integrating new services into the application should be done seamlessly, without the need for code changes. The first step towards that goal is to have a generic execution environment, that is capable of adapting to the services being modeled.

To clarify that more, the FlexMash execution engine is a Camunda engine instance, that is capable of executing the BPMN models that are obtained from the transformation phase steps 4 and 5 in Fig. 4.1 to provide the flexibility needed. In a BPMN model, each task has a service assigned to it to be called by the Camunda engine for the actual execution. That entails that for each new service added to FlexMash we need to add new code for that service.

Thus, the integration of FlexMash and Camunda required a generic execution service that adapts to all services, gathers the inputs from the previous nodes/tasks, and sends it to the next service to be implemented. This generic executor is responsible for calling the service registry to get the service information, then contacting the service itself with the information obtained from the registry as explained in Fig. 4.4.

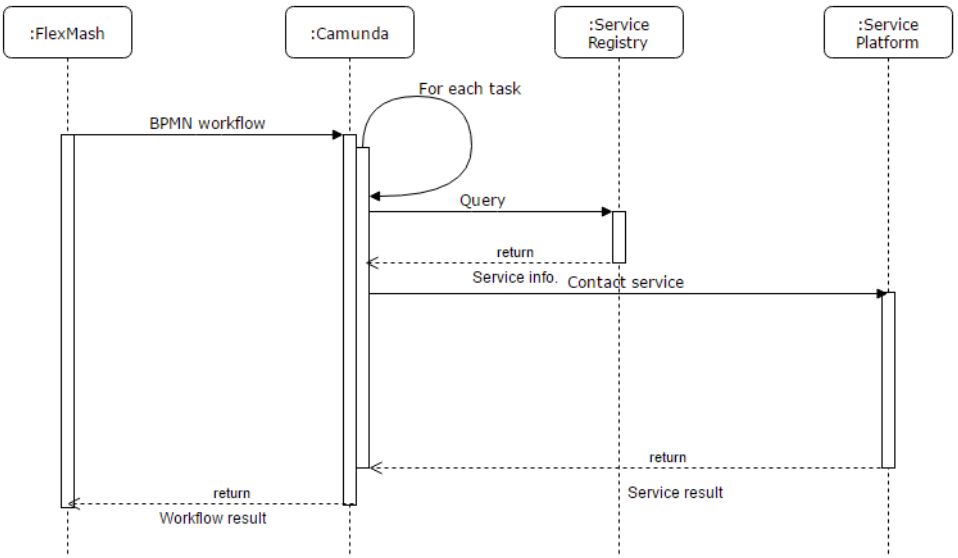


Figure 4.4: Execution flow

The service platform is built as shown in Fig. 4.3, so that it encapsulates the Consul operations and logic and the users can directly interact with it. It also adds more logic for the application such as authentication, service queries and service management. Using the features of the infrastructure, records are kept for the services registered with the Service Platform per user, as well as deletion rights. This also adds more control over the nodes running behind the Service Platform. For authentication purposes, the Service Platform requires users to acquire tokens for all future interactions, in order to be able to maintain the data of each user separately. This applies to all possible users, including the application FlexMash, which enables the integration as well as making services available for all FlexMash instances.

The Camunda engine integrates the intermediate results that were retrieved from the services into the execution using the job executor to store them in the process variables. For each executed model, an instance of the engine is created using the API provided, to ensure that each instance is autonomous and robust. This adds the possibility for the history of the executions and models generated to be maintained.

5 Implementation

In this section, the main concepts of this thesis with technical details are described, and a practical example is provided where the FlexMash is used in combination with the Service Platform. The example provides a concrete way to explain how the different components interact with each other, and will also highlight the technical details used and the reasons behind using them.

5.1 Implemented services

5.1.1 Filtration service

One of the services implemented throughout the course of this thesis is a service to apply filters to CSV data sets. The service takes as inputs the user's filtration criteria and the CSV data. The filtration criteria supports AND/OR logical operations. In the scenario modeled "sex = M AND address = U" is used. For both operations, each criteria and condition is evaluated separately, while keeping track of the indexes that match each condition.

The main difference is that for the AND operations, a match for each condition should be found. This means that the index of the entry should be found N times in the indexes list, where N is the number of conditions and in that case the entry is added to the result set.

As for the OR operation the same process is completed, with the exception of the last step. Here where the list of indexes that is constructed by matching each condition separately, is cleared of all duplicate entries, and the cleaned list is then returned as the result set.

5.1.2 Merging service

As mentioned earlier, the use cases for this thesis use the CSV data format. The merging service uses that same format, where it takes as inputs the merging conditions which also supports both logical operations as the filtration service, and two CSV datasets that will be merged. The service then creates a SQL DB instance in memory with both CSV data sets to be merged, represented as tables.

The goal behind this approach is to take advantage of the DBMS efficiency and make it take care of the merging logic. It is worth mentioning that the column names in both CSV data sets should be unique, otherwise the service will assign the conditions randomly, in case both data sets have the same column name. The results of this service are returned also in CSV format, after transforming the result SQL table into that format. An example condition can be (sex = sex AND reason = reason) to the two sets of data, which are similar in structure to the data in Table A.1. The result of the process is the resulting SQL table generated in CSV format.

5.2 Merge scenario

This example is to demonstrate the merge service, in which I used the datasets mentioned before to be merged. The scenario is depicted in Fig. 5.1a.

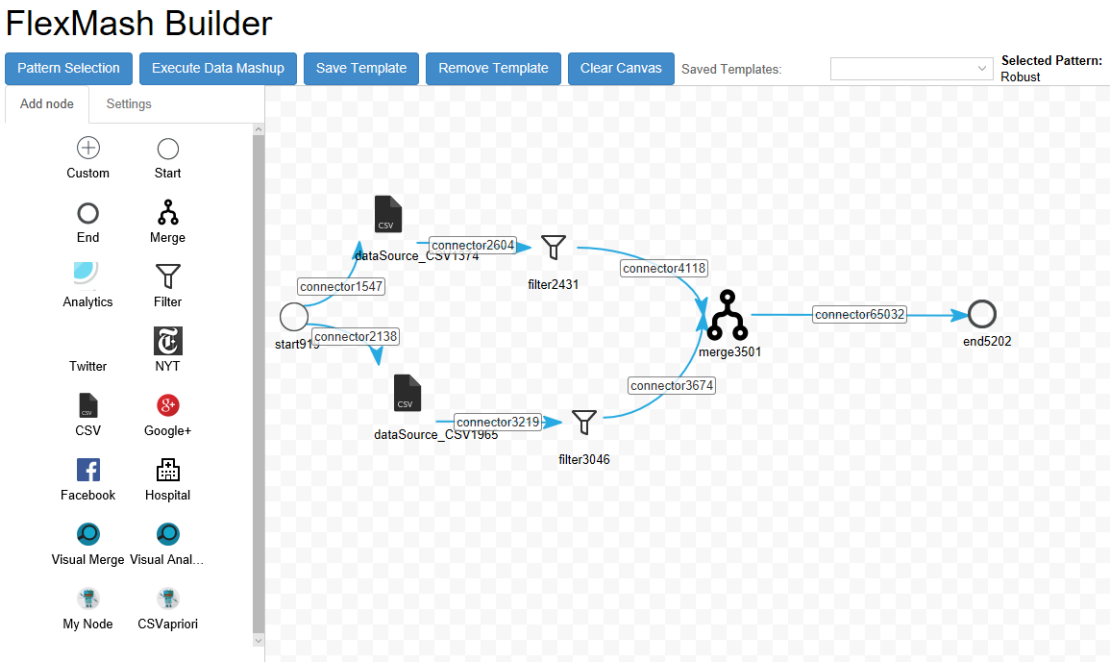
5.2.1 Sample data used

To run a practical example, a sample dataset[1] was obtained. A sample of the data is depicted in A.1, and consists of two CSV files to demonstrate the new services as well as the interaction between the components. The data were obtained in a survey of students' math and Portuguese language courses in secondary school. The data includes various social and gender studies about the students, which can be used to do different operations and analysis using the services implemented for this thesis.

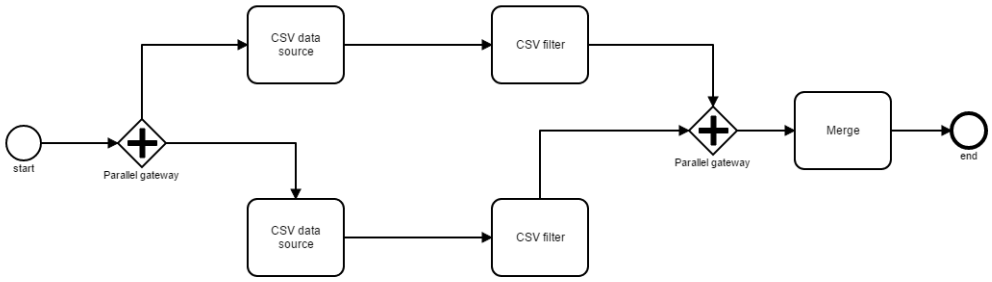
It is important to know that the objective of the scenarios demonstrated in this thesis, is to showcase the technical functionalities. The data provided is being used only for this purpose. That does not negate the fact that the services could be used for scenarios where the goal is processing the data for a different end goal.

[1] https://www.kaggle.com/uciml/studentalcoholconsumption

5.2.2 Model Transformation



(a) Merge example



(b) Result BPMN model

Figure 5.1: Merge example

The model in Fig. 5.1a is constructed by the FlexMash application user. It is then sent to the back end in JSON format that represents the details of that model. An example of that representation is depicted in Listing. A.1, where this JSON code represents the start node[2], as well as the transitions going out from it.

[2]I removed the styling elements for the sake of relevance to this context.

Looking at the Listing A.1, under the transition node, there are two data sources for CSV, which is the representation of Fig. 5.1a. The entire document is then transformed into a BPMN model, using the Camunda Java API. By traversing the entire model, creating the nodes that represent the model first, then creating the transitions between them. The result of that process is transforming an entire JSON model that consists of multiple nodes into the BPMN file that the Camunda engine can execute. The end result of that transformation is shown in Listing A.2[3].

When there are multiple outgoing transitions from one node, and multiple incoming transitions into a different node, the Camunda engine requires that Parallel Gateways (PG) are modeled before the multiple transitions, as depicted in Fig. 5.1b. The gateways in general and the PG in this case are used to model concurrency in a process. The functionality of the parallel gateway is based on the incoming and outgoing sequence flow(s):

- fork: all outgoing sequence flows are followed in parallel, creating one concurrent execution for each sequence flow.
- join: all concurrent executions arriving at the parallel gateway wait at the gateway until an execution has arrived for each of the incoming sequence flows. Then the process continues past the joining gateway.

The PG can have both fork and join behaviors, if there are multiple incoming and outgoing sequence flows for the same parallel gateway. In that case, the gateway will first join all incoming sequence flows, before splitting into multiple concurrent paths of executions. In our example, it allows forking into multiple paths of execution and joining multiple incoming paths of execution.

The user is not required to model those gateways, which is due to the fact that the transformation logic built takes care of that problem, without the need of the users' input. The transformation process first traverses the JSON representation in both directions from the start node to the end node and vice versa, while keeping track of all nodes represented, the outgoing transitions from each node with their targets, and the incoming transitions for each node with their sources.

Through the engine's Java API, the three lists are represented as objects of the relevant types. For each node with multiple incoming or outgoing transitions, this node is connected in the new model to a PG, which in turn takes all of the incoming or outgoing transitions as its own. The result of the process could be seen in the logical difference between both Listing A.1 and Listing A.2.

[3]This is a part of document for demonstration purposes for this thesis.

The code responsible for this replacement is depicted in the code snippit A.4, where lines starting from 9 are responsible for removing all the nodes with multiple outgoing transitions, and replaces them with the parallel gate in the list of sequence flows. The code traverses the model starting from the end node until the start node. Lines starting from 36 are responsible for removing all the nodes with multiple outgoing transitions, and replaces them with the parallel gate in the list of sequence flows. The traversal of the model in this case is done from the start node until the end node.

5.2.3 Execution

The Camunda engine takes care of the model execution, which is BPMN compliant as explained in the previous subsection. Each task has class attributes in the model, which are then used to specify which code functionalities should be used for this task. In our case, there are two different classes used, the first one being “CSVExe”. This is responsible for the reading of the local file. It loads the CSV files into the engine as process variables, that can be then used within the scope of the executable process.

The second class is “GenericExe”, which is the generic executor developed as part of this thesis’ extensions for FlexMash. It is responsible for getting results from the direct predecessors of the task, and making them available to the task being executed. This part is done automatically where all the inputs for a task, whether the user inputs to the task or outputs to the the predecessors, are collected by the generic executor.

The generic executor is responsible for the main integration part between FlexMash and the Service Platform. For each task being executed, it queries the service registry, using the service name, and gets the details of the service explained in previous sections. The main pieces of information used by the executor from the query result are the address of the service and the inputs of the service. Since it constructs the request to the service using those parameters, using JSON, the input names are used in requests made to the service. The flow of execution that is done for each task by the generic executor is:

1. Query the executable model and get the task’s predecessors
2. Get the outputs of the direct predecessors of the task being executed - if available -
3. Query the Service Platform using the task name
4. Contact the service using the information obtained from the query in step 2
5. Incorporate the result of the service into the process engine as output for the task being executed

In our example, due to the existence of the PGs in the executable mode, each branch of the transitions between the Start and the Merge tasks is executed concurrently, yet the execution of the merge task does not begin, until both branches are done with their execution, that way the engine is able to incorporate both inputs from the results of those branches, into the merge process through the generic executor. Without the modeling of the PG during the transformation phase, the engine would execute the Merge task twice, once for each branch.

5.3 Apriori scenario

A popular data mining approach is to find frequent itemsets from a transaction dataset and derive association rules. Finding frequent itemsets (itemsets with frequency larger than or equal to a user specified minimum support criteria). Once frequent itemsets are obtained, it is straightforward to generate association rules with confidence larger than or equal to a specified minimum confidence. The Apriori algorithm (Algorithm. A.1) is one of the popular algorithms, used in order to generate the desired association rules.

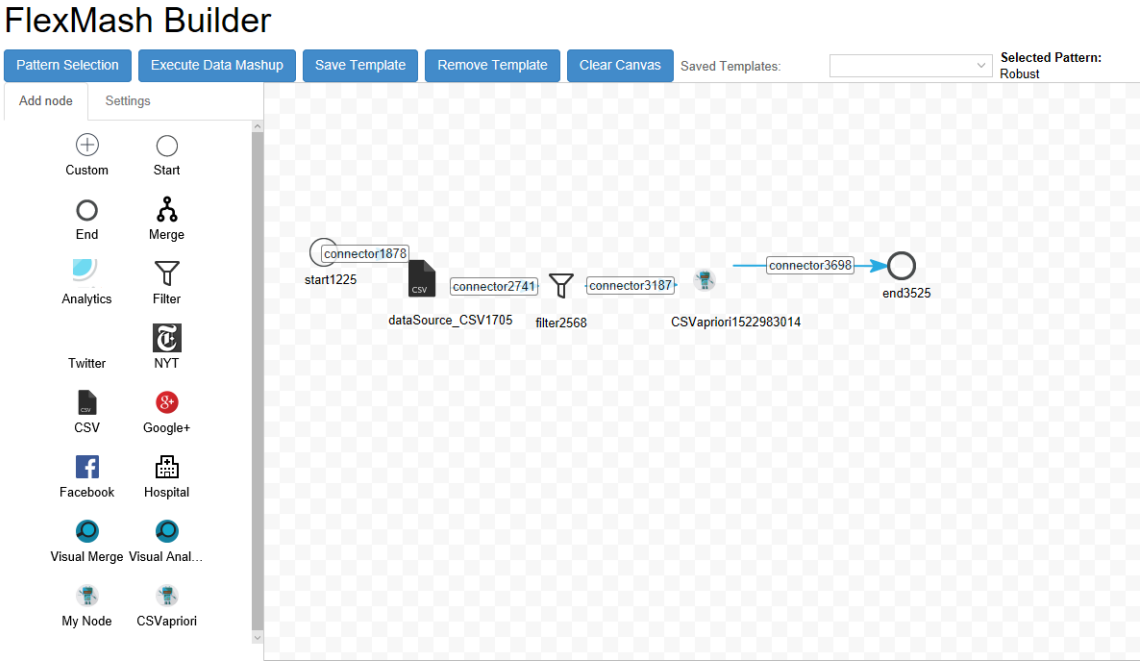


Figure 5.2: Apriori example

For the implementation of this service, I integrated the Weka[4] library into the service platform. It contains a collection of machine learning algorithms, that are used for data mining tasks. When it comes to the classification algorithms, it contains three different available algorithms, of which I integrated the Apriori algorithm. It provides different attributes which could be used to adjust the outcome, such as the minimum support, the minimum confidence, etc. According to [WFHP16], the implementation starts -assuming the default values are not changed- with a minimum support of 100% of the data items and decreases this in steps of 5% until there are at least 10 rules with the required minimum confidence of 0.9 or until the support has reached a lower bound of 10%, whichever occurs first.

For the scenario in Fig. 5.2, I used one from the data set mentioned in the previous scenario, in order to ensure that the service could be integrated with other services. As for the more relevant test, I used a supermarket dataset[5] which resulted in the association rules shown in A.3. The execution logic is the same as mentioned in the previous section.

5.4 Service Platform

The Service Platform offers a number of both POST and GET methods, available for users to interact with it. The methods offered comply with the definitions of both methods, where GET is idempotent, i.e, repeating the query does not have side-effects, while POST submits data to be processed, i.e, the POST requests have side-effects on the data maintained by the Service-platform. The available methods offered by the Service Platform are shown in Table 5.1 and Table 5.2. The aim of the methods and their separation between GET and POST, in addition to the Idempotence concept, is to make the Service Platform accessible for service consumers without the need for authentication. At the same time, it gives the service providers the ability to control their services, through the token authentication required for the POST methods.

When each of the methods is called, it calls the Consul engine and executes the requests. It also constructs the information received from the Consul engine into object oriented notions. The services which are available, are then returned to the requestors in JSON format,where they can map them to their own object structure. During the integration of the Service Platform with FlexMash, I added the service object structure. Thus, whenever the generic executor queries a service, it deals with an object instead of dealing with

[4]http://www.cs.waikato.ac.nz/ml/weka/
[5]http://storm.cis.fordham.edu/~gweiss/data-mining/datasets.html

the HTTP requests. The parameters for the object representing a service, are the same inputs needed for service registration, which are shown in Table. 5.1.

Method Signature	Functionality
getNewUserToken()	provides the user with a token for service admin interaction with the Service Platform
registerService(usertoken, port, healthcheck, ttl, servicename, serviceaddress, description, parameters, tags)	Allows providers to publish services on the Service Platform
deregisterService(usertoken, serviceid)	Allows providers to deregister services on the Service Platform

Table 5.1: POST methods

Method Signature	Functionality
getServicesWithTags(tags)	Users can query the Service Platform for services that are registered with certain tags.
getServiceName(servicename)	The service name is used to get a specific service's details.
getAllServices()	Returns all the services published on the Service Platform with their details.
getService(serviceid)	The service ID is used to get a specific service.

Table 5.2: GET methods

The aim of having the service platform as an intermediate layer between users and the Consul engine, instead of just using the capabilities of the engine, is to construct and control the logical interactions and use cases for the platform. It is also to be able to collect the data stored in the engine, to achieve an object oriented approach for communication, because the architectural approach for this thesis is not achieved directly by the Consul architecture. Data concerning the services are stored as services on the engine, while the data needed for the administration scenarios are stored in a

different data structure and a different place. To be able to unify both data sets into a common logic, the Service Platform is needed. I tried to achieve the best level of interaction simplicity throughout the thesis, which included choosing to work with JSON objects for simplicity, instead of web services for example. That way, users don't have to adapt to new WSDL files whenever they are changed. Simplicity also includes the object oriented approach, where service consumers can query the Service Platform and get the data they need, without the need to query different entities or to do more than one query to get the data related to the service they want to use.

6 Evaluation

In this chapter, I evaluate the implemented functionalities and concepts throughout this thesis.

Evaluation

According to Microsoft MSDN [CC06], there are four maturity levels - described below and shown in Fig. 6.1 - that we could classify SaaS into. Those levels are differentiated through different aspects like deployment customization, configurability, multi-tenant efficiency, and scalability.

Level 1 Ad-Hoc/Custom: At the first level of maturity, customers are able to customize their own versions that they are running on the application host's servers. Migrating a traditional non-networked or client-server application to this level requires the development effort to reduce operating costs.

Level 2 Configurable: this level provides better flexibility through configurable metadata, which enables many customers to use different instances of the same application code. This approach allows the vendor to meet the different needs of each customer through detailed configuration options, while simplifying maintenance of the common code base.

Level 3 Configurable, Multi-Tenant-Efficient: Adding multi-tenancy to the second level, so that a single program instance serves multiple customers. This approach enables potentially more efficient use of server resources without any apparent difference to the end user.

Level 4: Scalable, Configurable, Multi-Tenant-Efficient: explicit scalability features are added through a multi-tier architecture supporting a load-balanced farm of identical application instances, running on a variable number of servers. The system's capacity can be increased or decreased to match demand by adding or removing servers, without the need for any further alteration of application software architecture.

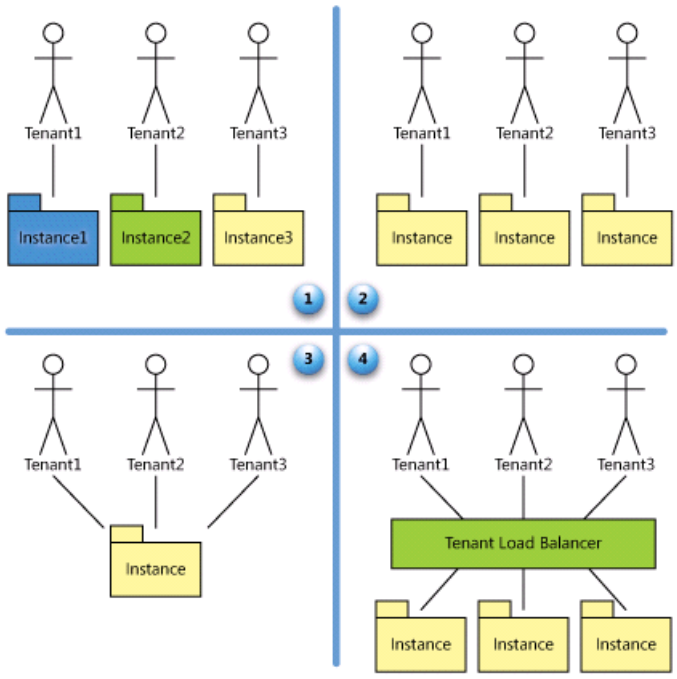


Figure 6.1: Maturity levels

The work done in this thesis, achieves many of the previously mentioned features to be able to achieve the goals explained in the first section. FlexMash allows the users to add their own services/nodes to the instances they are running, in order to model and process the scenarios that are relevant to them. Through the integration of FlexMash and the Service Platform, the integration of new services into the individual instances of the application is easier, which reduces the need for custom code changes depending on the user's needs.

It also helps with reducing the operating costs, since the registered services are hosted on the Service Platform in our case, which shifts the use of computation resources to the Service Platform instead of the users' instances. It also simplifies the maintenance of the services, since the code base is common and in one place, not hosted on each instance, which introduces centralization of the service maintenance.

As mentioned in the first section, the main goals for this thesis are to increase the flexibility of the application FlexMash, widen the borders of the possible data processing scenarios that the application can handle, while at the same time maintain the simplicity of the application, so that users with limited or no technical knowledge are able to use it.

Flexibility is achieved through different features, which includes the ability to integrate new services easily, and integration with the Service Platform which also enables the

generic execution of the services. Using the BPMN notation, instead of the BPEL notation that was previously used, also simplifies the required code maintenance. Mashup Plans are now transformed and executed seamlessly, with the ability to integrate more scenarios into the application.

7 Future work and Summary

7.1 Future work

There are many possibilities to enhance upon the work completed for this thesis. Integrating a database structure into the application, in order to keep track of the interactions between the users and the platform is an important step, because it will provide more data about the behaviors of both the services consumers and providers.

Such data can be used for analysis to discover new enhancement possibilities, through the identification of the use patterns of the application and the platform together. Since each Mashup plan executed in FlexMash is transformed into a BPMN model, there is also a possibility to record and keep track of the evolution and history of the models, which in turn can help users compare different models for the same scenario to find the optimal model for them.

Another important possibility is integrating a logging mechanism into the generic executor, to be able to analyze the overall execution of the model in a step by step style. Adding such a feature would help FlexMash users identify which services which can better suit their needs, since they would be able to integrate different services into the application, compare the results obtained from each of them, and decide which service serves the scenario being modeled better. That feature combined with keeping the history of the models, would help reduce the time needed to find the best model for the scenario using the best services available.

7.2 Summary

Data and application as services will keep growing in market share due to increasing demand in this area from both IT departments and users to simplify and cut down on IT budgets and complexity.

In this paper, an overview of the application FlexMash was provided, as well as the current state-of-the-art techniques it relies on, including data mashups, service discovery mechanisms, workflow engine, and BPMN.

The contributions of this thesis and possible future work that can be further developed from it would increase the maturity level of the FlexMash application. The main contributions of this thesis include shifting the "ownership" of the services provided to the Service Platform instead of the application instance. The application instance is then a terminal that communicates with the services provided on the Service Platform with no need to implement the services themselves on the instance.

This greatly improves the flexibility available for modeling data processing scenarios, even for users with relatively weak IT backgrounds. This allows a greater range of users to benefit from the application. There is also the possibilty to add more services from a wider range of service providers than before.

Furthermore, it reallocates the responsibility for the services infrastructure and management from the user to the service provider. By doing this, the computational costs are reduced, through hosting, providing and maintaining services in one place, instead of locally for each instance.

A Appendix

Algorithmus A.1 Apriori algorithm [WKQ+08]

F_l=(Frequent itemsets of cardinality 1);
for($k = 1; F_k \neq \phi; k++$) do begin
    C_{k+1} = apriori-gen(F_k); //New candidates
    for all transactions $t \in$ Database do begin
        C'_t = subset(C_{k+1}, t); //Candidates contained in t
        for all candidate $c \in C'_t$ do
            $c.count++$;
        end
        $F_{k+1} = \{C \in C_{k+1} \quad | c.count \geq$ minimum support $\}$
    end
end
Answer $\cup_k F_k$;

Listing A.1 Sample node JSON representation

```
{
"xy": [33, 200.00000762939453],
"name": "start1248",
"width": 40,
"description": "",
"transitions": [{
            "name": "connector2055"
      },
      "source": "start1248",
      "target": "dataSource_CSV1882"
      },{
            "name": "connector3011"
      },
      "sourceXY": [],
      "targetXY": [],
      "source": "start1248",
      "target": "dataSource_CSV2838"
      },
      "type": "start",
}
```

Listing A.2 Sample node BPMN format

```
<startEvent id="start1248" name="start1248">
  <outgoing>start1248-PGstart1248</outgoing>
</startEvent>
<serviceTask camunda:class="CSVExe" id="dataSource_CSV1882" name="CSV">
   <incoming>PGstart1248-dataSource_CSV1882</incoming>
   <outgoing>dataSource_CSV1882-filter3685</outgoing>
</serviceTask>
<serviceTask camunda:class="CSVExe" id="dataSource_CSV2838" name="CSV">
   <incoming>PGstart1248-dataSource_CSV2838</incoming>
  <outgoing>dataSource_CSV2838-filter4399</outgoing>
</serviceTask>
<serviceTask camunda:class="GenericExe" id="filter3685" name="">
  <incoming>dataSource_CSV1882-filter3685</incoming>
  <outgoing>filter3685-PGmerge4868</outgoing>
</serviceTask>
<serviceTask camunda:class="GenericExe" id="filter4399" name="">
  <incoming>dataSource_CSV2838-filter4399</incoming>
  <outgoing>filter4399-PGmerge4868</outgoing>
</serviceTask>
<serviceTask camunda:class="GenericExe" id="merge4868" name="merge">
  <incoming>PGmerge4868-merge4868</incoming>
  <outgoing>merge4868-end5705</outgoing>
</serviceTask>
<endEvent id="end5705" name="end5705">
  <incoming>merge4868-end5705</incoming>
</endEvent>
<parallelGateway camunda:async="false" id="PGstart1248" name="PGstart1248">
   <incoming>start1248-PGstart1248</incoming>
  <outgoing>PGstart1248-dataSource_CSV1882</outgoing>
   <outgoing>PGstart1248-dataSource_CSV2838</outgoing>
</parallelGateway>
 <sequenceFlow id="start1248-PGstart1248" sourceRef="start1248" targetRef="PGstart1248"/>
<parallelGateway camunda:async="false" id="PGmerge4868" name="PGmerge4868">
  <incoming>filter3685-PGmerge4868</incoming>
  <incoming>filter4399-PGmerge4868</incoming>
  <outgoing>PGmerge4868-merge4868</outgoing>
</parallelGateway>
```

Listing A.3 Sample Apriori output

```
Apriori
=======
Minimum support: 0.15 (694 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 17

Generated sets of large itemsets:
Size of set of large itemsets L(1): 44
Size of set of large itemsets L(2): 380
Size of set of large itemsets L(3): 910
Size of set of large itemsets L(4): 633
Size of set of large itemsets L(5): 105
Size of set of large itemsets L(6): 1
Best rules found:
1. biscuits=t frozen foods=t fruit=t total=high 788 ==> bread and cake=t 723
    <conf:(0.92)> lift:(1.27) lev:(0.03) [155] conv:(3.35)
2. baking needs=t biscuits=t fruit=t total=high 760 ==> bread and cake=t 696
    <conf:(0.92)> lift:(1.27) lev:(0.03) [149] conv:(3.28)
3. baking needs=t frozen foods=t fruit=t total=high 770 ==> bread and cake=t 705
    <conf:(0.92)> lift:(1.27) lev:(0.03) [150] conv:(3.27)
4. biscuits=t fruit=t vegetables=t total=high 815 ==> bread and cake=t 746 <conf:(0.92)>
    lift:(1.27) lev:(0.03) [159] conv:(3.26)
5. party snack foods=t fruit=t total=high 854 ==> bread and cake=t 779 <conf:(0.91)>
    lift:(1.27) lev:(0.04) [164] conv:(3.15)
6. biscuits=t frozen foods=t vegetables=t total=high 797 ==> bread and cake=t 725
    <conf:(0.91)> lift:(1.26) lev:(0.03) [151] conv:(3.06)
7. baking needs=t biscuits=t vegetables=t total=high 772 ==> bread and cake=t 701
    <conf:(0.91)> lift:(1.26) lev:(0.03) [145] conv:(3.01)
8. biscuits=t fruit=t total=high 954 ==> bread and cake=t 866 <conf:(0.91)> lift:(1.26)
    lev:(0.04) [179] conv:(3)
9. frozen foods=t fruit=t vegetables=t total=high 834 ==> bread and cake=t 757
    <conf:(0.91)> lift:(1.26) lev:(0.03) [156] conv:(3)
10. frozen foods=t fruit=t total=high 969 ==> bread and cake=t 877 <conf:(0.91)>
    lift:(1.26) lev:(0.04) [179] conv:(2.92)
```

school	sex	age	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	reason
GP	F	18	GT3	A	4	4	at_home	teacher	course
GP	F	17	GT3	T	1	1	at_home	other	course
GP	F	15	LE3	T	1	1	at_home	other	other
GP	F	15	GT3	T	4	2	health	services	home
GP	F	16	GT3	T	3	3	other	other	reputation
GP	M	17	GT3	T	3	2	services	services	course
GP	M	16	LE3	T	4	3	health	other	home
GP	M	15	GT3	T	4	3	teacher	other	reputation
GP	M	15	GT3	T	4	4	health	health	other
GP	M	16	LE3	T	4	2	teacher	other	course
GP	M	16	LE3	T	2	2	other	other	reputation
GP	F	15	GT3	T	2	4	services	health	course
GP	F	16	GT3	T	2	2	services	services	home
GP	M	15	GT3	T	2	2	other	other	home
GP	M	15	GT3	T	4	2	health	services	other
GP	M	16	LE3	A	3	4	services	other	home
GP	M	16	GT3	T	4	4	teacher	teacher	home
GP	M	15	GT3	T	4	4	health	services	home
GP	M	15	GT3	T	4	4	services	services	reputation
GP	M	16	GT3	T	3	3	services	other	home
GP	F	17	GT3	T	2	4	services	services	reputation
MS	F	17	GT3	T	1	2	other	other	course
MS	F	18	LE3	T	4	4	other	other	reputation
MS	F	18	GT3	T	1	1	other	other	home
MS	F	20	GT3	T	4	2	health	other	course
MS	F	18	LE3	T	4	4	teacher	services	course
MS	F	18	GT3	T	3	3	other	other	home
MS	F	17	GT3	T	3	1	at_home	other	reputation
MS	M	18	GT3	T	4	4	teacher	teacher	home
MS	M	18	GT3	T	2	1	other	other	other
MS	M	17	GT3	T	2	3	other	services	home

Table A.1: Sample data

Listing A.4 Parallel-Gateway handling

```
public void replaceValues(Map<String, ArrayList<String>> outTransitionMap,
  Map<String, ArrayList<String>> inTransitionMap,
  BPMNmodel BPMNWorkFlow) {

 // Iterator for source nodes
 ArrayList<String> replaceKeys = new ArrayList<>();
 Map<String, ArrayList<String>> outTransitionMapReplaced = new HashMap<String,
     ArrayList<String>>();

 for (Iterator<Entry<String, ArrayList<String>>> outIter = outTransitionMap
  .entrySet().iterator(); outIter.hasNext();) {

  // oEntry <Source node, list of target nodes>
  Map.Entry<String, ArrayList<String>> oEntry = (Map.Entry<String, ArrayList<String>>)
      outIter
   .next();

  if (oEntry.getValue().size() > 1 && !oEntry.getKey().startsWith("PG")) {
   ParallelGateway parallel = BPMNWorkFlow.createElement(
    BPMNWorkFlow.MainProcess, "PG" + oEntry.getKey(),
    ParallelGateway.class);
   parallel.setName("PG" + oEntry.getKey());
   parallel.setCamundaAsync(false);
   BPMNWorkFlow.createSequenceFlow(
    BPMNWorkFlow.MainProcess, BPMNWorkFlow.ModelInstance
     .getModelElementById(oEntry.getKey()),
    parallel);
   outTransitionMapReplaced.put(parallel.getName(),
    oEntry.getValue());
   replaceKeys.add(oEntry.getKey());

  }
 }
 replaceKeys.forEach((k) -> outTransitionMap.remove(k));
 outTransitionMapReplaced.forEach((k, v) -> outTransitionMap.put(k, v));
 replaceKeys.clear();
```

```
35
36   for (Iterator<Entry<String, ArrayList<String>>> inIter = inTransitionMap
37     .entrySet().iterator(); inIter.hasNext();) {
38    Map.Entry<String, ArrayList<String>> iEntry = (Map.Entry<String, ArrayList<String>>)
          inIter
39     .next();
40    if (iEntry.getValue().size() > 1) {
41
42     ParallelGateway parallel = BPMNWorkFlow.createElement(
43       BPMNWorkFlow.MainProcess, "PG" + iEntry.getKey(),
44       ParallelGateway.class);
45     parallel.setName("PG" + iEntry.getKey());
46     parallel.setCamundaAsync(false);
47     BPMNWorkFlow.createSequenceFlow(BPMNWorkFlow.MainProcess,
48       parallel, BPMNWorkFlow.ModelInstance
49         .getModelElementById(iEntry.getKey()));
50
51     for (Iterator<String> sourceNodes = iEntry.getValue()
52       .iterator(); sourceNodes.hasNext();) {
53
54      String sourceNode = sourceNodes.next();
55
56      outTransitionMap.get(sourceNode).add(parallel.getName());
57      outTransitionMap.get(sourceNode).remove(iEntry.getKey());
58     }
59
60    }
61   }
62
63  }
```

Bibliography

[All16] T. Allweyer. *BPMN 2.0: introduction to the standard for business process modeling*. BoD–Books on Demand, 2016 (cit. on p. 29).

[AM07] E. Al-Masri, Q. H. Mahmoud. "Discovering the best web service." In: *Proceedings of the 16th international conference on World Wide Web*. ACM. 2007, pp. 1257–1258 (cit. on p. 25).

[AZ05] P. Avgeriou, U. Zdun. "Architectural patterns revisited-a pattern language." In: (2005) (cit. on p. 19).

[CC06] F. Chong, G. Carraro. "Architecture strategies for catching the long tail." In: *MSDN Library, Microsoft Corporation* (2006), pp. 9–10 (cit. on p. 61).

[CJCR04] J. Colgrave, K. Januszewski, L. Clément, T. Rogers. "Using wsdl in a uddi registry, version 2.0. 2." In: *Technical note, OASIS* (2004) (cit. on pp. 23, 24).

[CLPZ] K. S. Candan, W.-S. Li, T. Phan, M. Zhou. "Frontiers in Information and Software as Services." In: () (cit. on p. 45).

[DHPB09] G. Di Lorenzo, H. Hacid, H.-y. Paik, B. Benatallah. "Data Integration in Mashups." In: *SIGMOD Record* 38.1 (2009), p. 59 (cit. on pp. 35, 36).

[Dob03] E.-E. Doberkat. "Pipelines: Modelling a software architecture through relations." In: *Acta Informatica* 40.1 (2003), pp. 37–79 (cit. on pp. 19, 20).

[Dra01] V. Draluk. "Discovering Web Services: An Overview." In: *VLDB*. 2001, pp. 637–640 (cit. on pp. 23, 24).

[DSW97] K. Decker, K. Sycara, M. Williamson. "Middle-agents for the internet." In: *IJCAI (1)*. 1997, pp. 578–583 (cit. on p. 25).

[EAA+04] M. Endrei, J. Ang, A. Arsanjani, S. Chua, P. Comte, P. Krogdahl, M. Luo, T. Newling. *Patterns: service-oriented architecture and web services*. IBM Corporation, International Technical Support Organization, 2004 (cit. on pp. 21, 22, 28).

[FHA+99] G. Fox, T. Haupt, E. Akarsu, A. Kalinichenko, K.-S. Kim, P. Sheethalnath, C.-H. Youn. “The gateway system: uniform Web based access to remote resources.” In: *Proceedings of the ACM 1999 conference on Java Grande*. ACM. 1999, pp. 1–7 (cit. on p. 25).

[FO09] E. B. Fernandez, J. L. Ortega-Arjona. “The secure pipes and filters pattern.” In: *Database and Expert Systems Application, 2009. DEXA'09. 20th International Workshop on*. IEEE. 2009, pp. 181–185 (cit. on pp. 20, 21).

[GK04] B. Gröne, F. Keller. *Conceptual Architecture Patterns: FMC-based Representation*. Universitätsverlag, 2004 (cit. on p. 20).

[GPST04] J. Garofalakis, Y. Panagis, E. Sakkopoulos, A. Tsakalidis. “Web service discovery mechanisms: Looking for a needle in a haystack.” In: *International Workshop on Web Engineering*. Vol. 38. 2004 (cit. on p. 25).

[GPST06] J. Garofalakis, Y. Panagis, E. Sakkopoulos, A. Tsakalidis. “Contemporary web service discovery mechanisms.” In: *J. Web Eng.* 5.3 (2006), pp. 265–290 (cit. on pp. 24, 25).

[GS93] D. Garlan, M. Shaw. “An introduction to software architecture.” In: *Advances in software engineering and knowledge engineering* 1.3.4 (1993) (cit. on pp. 15, 19, 24).

[HB16] P. Hirmer, M. Behringer. “FlexMash 2.0–Flexible Modeling and Execution of Data Mashups.” In: *International Rapid Mashup Challenge*. Springer. 2016, pp. 10–29 (cit. on pp. 15, 16, 33, 34).

[HM16] P. Hirmer, B. Mitschang. “FlexMash–Flexible Data Mashups Based on Pattern-Based Model Transformation.” In: *Rapid Mashup Development Tools*. Springer International Publishing, 2016, pp. 12–30 (cit. on pp. 15, 16, 33, 34).

[HNP+11] A. Harth, B. Norton, A. Polleres, B. Sapkota, S. Speiser, S. Stadtmüller, O. Suominen. “Towards Uniform Access to Web Data and Services.” In: (2011) (cit. on p. 25).

[HRWM15] P. Hirmer, P. Reimann, M. Wieland, B. Mitschang. “Extended Techniques for Flexible Modeling and Execution of Data Mashups.” In: *DATA*. 2015, pp. 111–122 (cit. on pp. 15, 16, 30, 33–35, 42).

[KAB+04] M. Keen, A. Acharya, S. Bishop, A. Hopkins, S. Milinski, C. Nott, R. Robinson, J. Adams, P. Verschueren. “Patterns: Implementing an SOA using an enterprise service bus.” In: *IBM Redbooks* 336 (2004) (cit. on pp. 20, 28).

[LHSL07] X. Liu, Y. Hui, W. Sun, H. Liang. “Towards service composition based on mashup.” In: *Services, 2007 IEEE Congress on*. IEEE. 2007, pp. 332–339 (cit. on p. 37).

[Lie13] S. Lie. “Enabling the compatible evolution of services based on a cloud-enabled ESB solution.” MA thesis. 2013 (cit. on pp. 26, 27).

[Men07] F. Menge. “Enterprise service bus.” In: *Free and open source software conference*. Vol. 2. 2007, pp. 1–6 (cit. on p. 26).

[MK10] S. Mallick, D. Kushwaha. “An efficient web service discovery architecture.” In: *International Journal of Computer Applications* 3.12 (2010), pp. 1–5 (cit. on p. 25).

[MKMG97] R. T. Monroe, A. Kompanek, R. Melton, D. Garlan. “Architectural styles, design patterns, and objects.” In: *IEEE software* 14.1 (1997), pp. 43–52 (cit. on p. 20).

[MRG08] E. M. Maximilien, A. Ranabahu, K. Gomadam. “An online platform for web apis and service mashups.” In: *IEEE Internet Computing* 12.5 (2008) (cit. on p. 45).

[Muh12] D. Muhler. “Extending an open source enterprise service bus for multi-tenancy support focusing on administration and management.” MA thesis. 2012 (cit. on p. 27).

[PH07] M. P. Papazoglou, W.-J. Heuvel. “Service oriented architectures: approaches, technologies and research issues.” In: *The VLDB Journal—The International Journal on Very Large Data Bases* 16.3 (2007), pp. 389–415 (cit. on pp. 21, 22, 25–28).

[PR99] J. Philipps, B. Rumpe. “Refinement of pipe-and-filter architectures.” In: *FM’99—Formal Methods* (1999), pp. 708–709 (cit. on p. 20).

[Ran03] S. Ran. “A model for web services discovery with QoS.” In: *ACM Sigecom exchanges* 4.1 (2003), pp. 1–10 (cit. on p. 21).

[Rec08] J. C. Recker. “BPMN modeling–who, where, how and why.” In: *BPTrends* 5.3 (2008), pp. 1–8 (cit. on p. 29).

[RIRG06] J. C. Recker, M. Indulska, M. Rosemann, P. Green. “How good is BPMN really? Insights from theory and practice.” In: (2006) (cit. on p. 28).

[SKWL99] K. Sycara, M. Klusch, S. Widoff, J. Lu. “Dynamic service matchmaking among agents in open information environments.” In: (1999) (cit. on p. 25).

[SPAS03] K. Sycara, M. Paolucci, A. Ankolekar, N. Srinivasan. “Automated discovery, interaction and composition of semantic web services.” In: *Web Semantics: Science, Services and Agents on the World Wide Web* 1.1 (2003), pp. 27–46 (cit. on pp. 23, 24).

[SRAW03] A. ShaikhAli, O. F. Rana, R. Al-Ali, D. W. Walker. “Uddie: An extended registry for web services.” In: *Applications and the Internet Workshops, 2003. Proceedings. 2003 Symposium on*. IEEE. 2003, pp. 85–89 (cit. on pp. 23, 24).

[TAR07] A. Thor, D. Aumueller, E. Rahm. “Data Integration Support for Mashups.” In: (2007) (cit. on p. 35).

[TSB10] W.-T. Tsai, X. Sun, J. Balasooriya. “Service-oriented cloud computing architecture.” In: *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*. IEEE. 2010, pp. 684–689 (cit. on p. 44).

[WAD+] P. Wohed, W. M. van der Aalst, M. Dumas, A. H. ter Hofstede, N. Russell. “Pattern-based Analysis of BPMN.” In: () (cit. on p. 29).

[WFHP16] I. H. Witten, E. Frank, M. A. Hall, C. J. Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016 (cit. on p. 57).

[Whi04] S. A. White. “Introduction to BPMN.” In: *IBM Cooperation* 2.0 (2004), p. 0 (cit. on p. 28).

[WKQ+08] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip, et al. “Top 10 algorithms in data mining.” In: *Knowledge and information systems* 14.1 (2008), pp. 1–37 (cit. on p. 69).

[YBCD08] J. Yu, B. Benatallah, F. Casati, F. Daniel. “Understanding Mashup Development.” In: *IEEE Internet Computing* 12.5 (2008), p. 44 (cit. on pp. 16, 29, 30, 33).

[ZLCC02] L.-J. Zhang, H. Li, H. Chang, T. Chao. “XML-based advanced UDDI search mechanism for B2B integration.” In: *Advanced Issues of E-Commerce and Web-Based Information Systems, 2002.(WECWIS 2002). Proceedings. Fourth IEEE International Workshop on*. IEEE. 2002, p. 4 (cit. on p. 23).

All links were last followed on July, 2017.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature