FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Software Architecture Evolution of Collective Intelligence Systems into System-of-Systems Farm Platforms

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Software Engineering and Internet Computing

eingereicht von

Elisabeth Pilz, BSc
Matrikelnummer 01225231

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr. Mag. Stefan Biffl
Mitwirkung: Angelika Musil
Jürgen Musil

Wien, 2. April 2019

Elisabeth Pilz	Stefan Biffl

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Software Architecture Evolution of Collective Intelligence Systems into System-of-Systems Farm Platforms

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Software Engineering and Internet Computing

by

Elisabeth Pilz, BSc
Registration Number 01225231

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr. Mag. Stefan Biffl
Assistance: Angelika Musil
Jürgen Musil

Vienna, 2nd April, 2019

Elisabeth Pilz

Stefan Biffl

Erklärung zur Verfassung der Arbeit

Elisabeth Pilz, BSc
Luegstraße 13, 3340 Waidhofen/Ybbs

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 2. April 2019

Elisabeth Pilz

Acknowledgements

This work was conducted in the context of the Christian Doppler Laboratory for Security and Quality Improvement in the Production System Lifecycle (CDL-SQI), Institute of Information Systems Engineering, TU Wien. The financial support by the Christian Doppler Research Association, the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged.

Many thanks to Prof. Biffl for his supervision and his constructive feedback to improve my thesis. Another special thanks goes to Angelika Musil and Jürgen Musil who provided this interesting research topic and gave me helpful suggestions and permanent feedback.

Special thanks to my family and my friends who supported and motivated me during my studies. Thank you for all your warm words and for making my life bright and sunny.

Kurzfassung

Collective Intelligence Systeme (CIS) sind webbasierte Plattformen, die das Wissen von vernetzten Personengruppen nutzen. Ein CIS lebt von seinen Benutzern, die Inhalte zu einem gemeinsam global genutzten Informationsraum beitragen und dadurch auch ihr Wissen erweitern können. Ein CIS kann für verschiedene Arten von Wissensbasen nützlich sein, die rund um ein bestimmtes Thema gruppiert und organisiert sind. Daher möchten Betreiber solcher Plattformen die Systemfunktionen für Anwendungen mit ähnlichen Strukturen wiederverwenden. Dies führt jedoch zu Kopien von Instanzen, die nur mit unterschiedlichem Wissen gefüllt sind, und sich mehr oder weniger unabhängig vom ursprünglichen System weiterentwickeln. Dieser Ansatz führt jedoch zu mehreren Herausforderungen und Einschränkungen, wie z. B. die komplexe Weiterentwicklung der Systemtechnologien, ein hoher Verwaltungsaufwand, verschiedene Benutzerkonten für eine Person und keine instanzübergreifende Verknüpfung von Wissen.

Um diese Einschränkungen zu beheben, wurde in dieser Arbeit ein neuer Architekturansatz entwickelt, die CIS Farmarchitektur. Bei diesem Architekturansatz sind mehrere CIS Instanzen in eine Systemumgebung eingebettet. Dieser Ansatz bietet Funktionen zum Erstellen, Bereitstellen und Verwalten unabhängiger Farmentitäten. Im Kontext einer CIS Farm wir eine CIS Instanz als Farmentität bezeichnet. Der Schwerpunkt der Forschungsfragen lag auf den Merkmalen, der Entwicklung und der Evaluierung des CIS Farm Ansatzes. Die Hauptbeiträge dieser Arbeit gliedern sich in die Definition der architektonischen Prinzipien, die Beschreibung des CIS Farm Architektur Designs und die Entwicklung eines Evolutionsprozesses von einem CIS zu einer CIS Farm Architektur.

Die Beiträge dieser Arbeit wurden an einem praktischen Beispiel angewandt - der Glossar Plattform. Die Glossar Plattform ist ein CIS, bei dem mehrere Benutzer geografisch unabhängig zusammenarbeiten, indem sie eine gemeinsam genutzte Online-Plattform verwenden, um eine Ansammlung von Begriffen mit zugehörigen Definitionen zu verwalten. Nach der Implementierung wurde die Glossary Farm durch eine Anwenderstudie evaluiert. Die Teilnehmer stimmten der Aussage zu, dass die Glossary Farm den Aufwand für die Erstellung, Verwendung und Verwaltung eines Glossars reduziert, und weitere wichtige Funktionen bereitstellt.

Zusammenfassend bieten die Beiträge dieser Arbeit eine gute Grundlage für die Implementierung einer CIS Farm. Eine CIS Farm hat den Vorteil, dass mehrere Instanzen in eine Systemumgebung integriert sind und das Wissen darüber verknüpft werden kann. Darüber hinaus kann der Administrationsaufwand drastisch reduziert werden.

Abstract

Nowadays, the interest in networked and collectively created knowledge bases grows rapidly. Collective intelligence systems (CIS) are web-based social platforms that use the knowledge of connected groups of people. A CIS thrives from its users, who contribute content to a globally shared information space. Each human actor benefits from the available shared information and can expand her knowledge. CIS may be useful with different kinds of knowledge base content, grouped and organized around a specific topic. Therefore operators of these platforms want to reuse the system capabilities for knowledge with similar structures. Today, this need leads to instance clones, where simply a copy of the CIS is filled with new knowledge and evolves more or less independently from the original system. This approach has several challenges and limitations, such as complex system technology evolution, impossible data analysis across knowledge bases, high administration effort, various accounts for one person and no cross-instance linking of knowledge.

To resolve these limitations, we designed a new architectural approach, the CIS farm architecture. Within a CIS farm architecture, several CIS instances are embedded into one system environment. Therefore this approach provides functions to simply create, deploy and administrate individual and independent farm entities. In the context of a CIS farm we call a CIS instance a farm entity. The focus of the research questions is on the characteristics, the evolution and the evaluation of the CIS farm approach. The main contributions of this thesis are the definition of the architectural principles, the description of the CIS farm architecture design and the development of an evolution process from a CIS to a CIS farm architecture.

The contributions of this thesis were applied using a practical example - the Glossary Platform. The Glossary Platform is a CIS where multiple users collaborate geographically independent by using a shared online platform to administrate a collection of terms with associated definitions. Afterward, the Glossary Farm was evaluated by conducting a user study. The participants agreed with the statement that the Glossary Farm could reduce the effort of creating, using and maintaining a glossary and provides some important features.

In summary, the contributions of this thesis provide a suitable foundation for the implementation of a CIS farm. A CIS farm has the advantage that several instances are integrated into one system environment and the knowledge of these can be cross-linked. Furthermore, the administration effort can be reduced dramatically.

Contents

CHAPTER 1

Introduction

In the last decades new forms of knowledge creation and sharing have emerged. The new forms of web-based social platforms have the capability to use the collective intelligence of connected groups of people. For instance, social networks enhance the collective knowledge every time users publish new content [1]. All these systems can be regarded as Collective Intelligence Systems (CIS), which aggregate and distribute information and content among their user base in an efficient way [2]. A key characteristic of CIS is that the shared information is focused around a certain topic. Therefore it is possible that various instances of one system exist for different communities. Examples for CIS are wikis and the online encyclopedia *Wikipedia*[1], review and rating platforms like *Yelp*[2] or question and answer platforms like *StackOverflow*[3].

Although collective intelligence already existed before the Internet, the Internet opened the possibility to gather information from thousands or even millions of people [3]. Each person has different knowledge and information of specific topics. A CIS is formed when this knowledge is collected centrally on a platform and therein shared with a user community [4]. CIS may be useful with different kinds of knowledge base content, grouped and organized around a specific topic. Therefore operators of these platforms want to reuse the system capabilities for knowledge with similar structures. Today, this need leads to *instance clones*, where simply a copy of the CIS is filled with new knowledge and evolves more or less independently from the original system. However, this approach has two major challenges, which are (1) system technology evolution [5], i.e., fixing technical defects and capability evolution in a family of systems and (2) integrated data analysis across knowledge bases.

[1] `https://www.wikipedia.org`, last visited at 03.01.2019
[2] `https://www.yelp.com`, last visited at 03.01.2019
[3] `https://www.stackoverflow.com`, last visited at 03.01.2019

1.1 Motivating Example

To motivate the need we introduce the *Glossary Platform* as an example, which will be used in the rest of this thesis. A glossary is a collection of terms with associated definitions, either to a particular domain or to a specific project. For example, a glossary is used in research collaborations to reduce terminological misunderstandings and inconsistencies by providing a single agreed definition for a particular term. The Glossary Platform is a CIS where multiple users collaborate geographically independent by using a shared online platform. This online platform is used to collect and share terms across all scientists. Each scientist benefits from the available shared information from others and is encouraged to contribute her own knowledge [4].

The Glossary Platform is a prototype developed at TU Wien and is already in use for projects of the Institute of Information Systems Engineering, e.g., the *CDL-SQI Glossary*[4] that is used in a research project. Currently, a separate glossary instance for each research collaboration is created and deployed. This leads to several limitations, such as:

- **Large number of instances**
  A new Glossary instance has to be created and deployed for every research project or community. As a result, one can quickly lose the overview of all available Glossary instances. There is no central point where all Glossaries are listed and described. A new Glossary instance can only be discovered when other researchers communicate this or invite a scientist to contribute her knowledge.

- **Many accounts for one researcher**
  If a researcher wants to contribute in several glossaries, she needs separate accounts for each instance. Actually, this leads only to a duplication of data, because the same account (for one person) has to be created several times.

- **No cross-instance linking of knowledge**
  Currently, it is not possible to link terms across different instances. A Glossary instance is a self-contained system without any external interfaces. Therefore, it is not possible to network glossaries and link knowledge across various instances. So it is impossible to collaborate across multiple domains.

- **High administration effort**
  The maintenance of all the Glossary instances results in a significant effort for the administrator. All instances must be maintained and kept up-to-date (product updates and bug-fixes). Furthermore, the system administrator has to monitor all instances to see, if they are still online.

[4] `https://glossary-cdl-sqi.herokuapp.com`, last visited at 15.06.2018

- **Limited data analysis**
  An analysis of the data is only possible per instance, an overlapping analysis across various glossaries is not possible. Statistics about several instances must be created manually if necessary. Furthermore, the activities of a user can only be tracked per instance. Personalized summaries about multiple instances are therefore not possible.

Figure 1.1 outlines the current limitations of the existing Glossary Platform. This view illustrates some of the limitations as mentioned above, e.g., many instances, several accounts for one researcher and no cross-instance linking of knowledge.

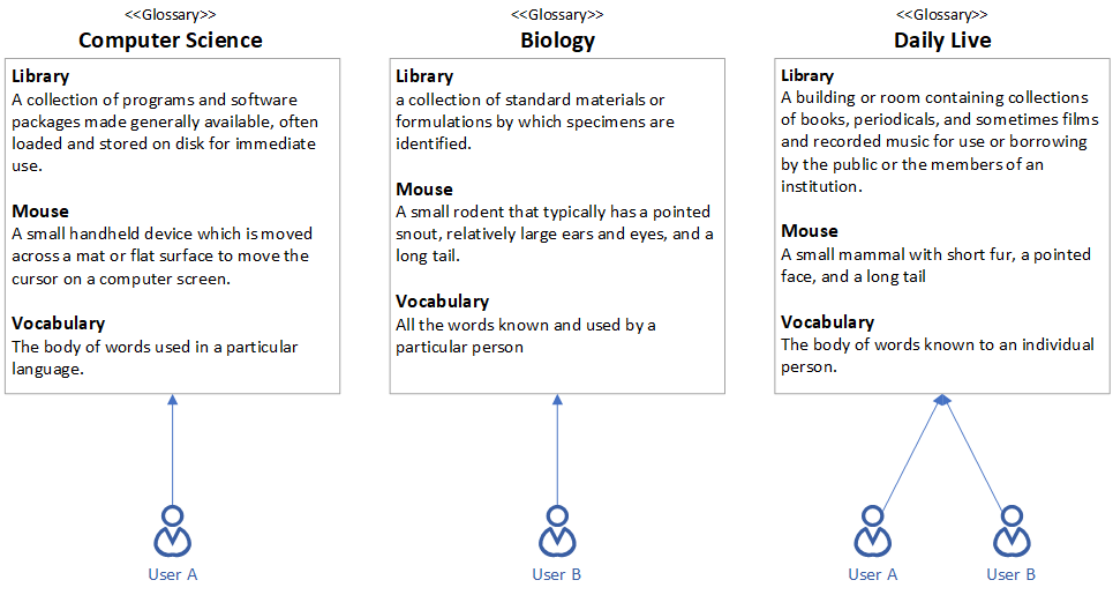

Figure 1.1: Limitations of the current Glossary Platform

1.2 Problem Statement

In the context of the Glossary Platform there are still more challenges than just system technology evolution and integrated data analysis. Additional drawbacks are the number of accounts for one researcher or that no cross-instance linking of knowledge is possible. In the following, various approaches are discussed how to create a network of glossaries. Every approach has its own advantages and disadvantages.

Instance clones

Instance clones are mostly used to make copies of platforms and to fill than with new, different knowledge. With this approach the administration effort increases enormously, since fixing technical defects or capability evolution must be carried out separately for each instance. Other disadvantages have already been discussed in detail in the previous section. The main advantage of this approach is that a new Glossary can be created without any implementation effort. Only a new instance needs to be created and deployed.

All in one

In principle, it would be possible to collect the whole knowledge in one Glossary. Some of the limitations would be annulled, but other new problems would be created. This approach possesses only one instance. Thereby the administration effort can be reduced. Besides, scientists have only one account, and the data analysis is possible in greater detail. New emerging problems incluce (1) no differentiation of communities possible and (2) the loss of clarity. Moreover, some terms in different domains have another meaning, so a wide variety of definitions for one term exists. As a result, the comprehensibility is lost since nobody knows which definition of a term belongs to which domain. In conclusion, this approach is unsuitable because the emerging problems are too serious.

Virtualization

An appropriate solution for the maintenance issue would be virtualization. The advantages of virtualization are (1) resource sharing, i.e., the underlying hardware can be shared between all virtual machines (VMs), (2) isolation, i.e., applications can be executed in different virtual machines, (3) dynamical provision of a VM and (4) dynamic moving of a VM among different servers [6]. There are several levels of virtualization, the most appropriate type for the Glossary environment scenario would be the Operating System-level virtualization or container-based virtualization. Containers are isolated, but share the same operating system and kernel. The advantage of this is mainly the performance since near-native speeds can be achieved [7].

Figure 1.2 illustrates the container-based virtualization of the Glossary environment. Each container encloses a glossary and isolates the application. Therefore, updates or bug fixes in the application can be automated via routines. New containers (glossaries) can be easily created with the use of a template. With virtualization the cost of maintenance can be dramatically reduced. However, limitations arise in the integrated data analysis and sharing of data across different containers (knowledge bases). The container-based approach allows only data analysis per Glossary or the exchange of data between different instances via a dedicated interface (no cross-instance linking possible). In summary, the virtualization approach eliminates not all limitations.

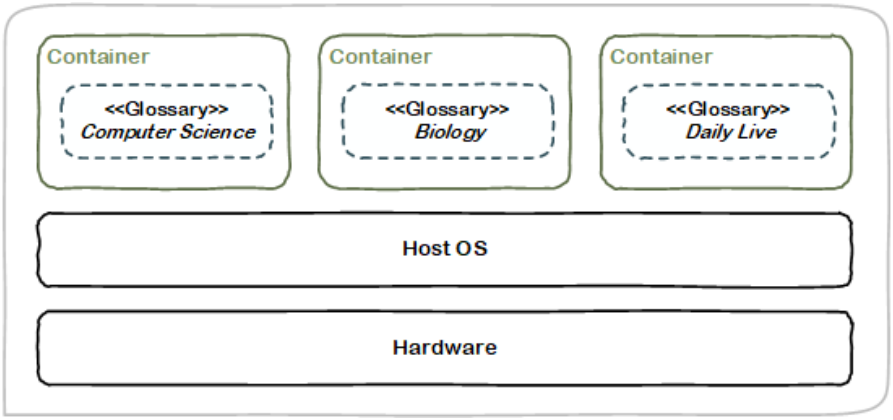

Figure 1.2: Container-based virtualization of the Glossary environment

Because the previously described approaches eliminate not all limitations, it may be worth considering to investigate a new architecture approach for such CIS. The new approach should offer a possibility to integrate all Glossary instances in one environment and try to eliminate all outlined issues.

1.3 Contributions of this Thesis

The existing approaches are not sufficient to create a network of CIS. Possibilities which are well-known at the moment have too many disadvantages and meet not the requirements of a network of identical systems. A kind of system-of-systems approach is needed, where the individual glossaries are integrated into one enclosing system. All glossaries in this union should have the same functions, but filled with different knowledge from varying user communities.

An initial literature study identified two related, but different concepts for this union. Firstly, *System of Systems* (SoS), where independent software systems are working together to fulfill a mission that no system could provide alone [8]. Secondly, *Multi-tenancy Systems*, where tenants share resources such as hardware and software, without necessarily sharing data [9]. However, these concepts only provide ideas for a new suitable architecture design, because they do not meet the criteria for a network of CIS. An architecture design approach is needed, where (1) on a single, shared platform multiple instances can be hosted, (2) administrators can effortlessly create and maintain instances, (3) knowledge can be linked across various instances, and (4) an integrated data analysis across knowledge bases is possible.

A promising approach presents the architecture concept of *WikiFarms*, where a ”farm” grows wikis [10]. A farm platform is a sort of system-of-systems, where several CIS are integrated into one application. A single CIS in the farm design is called *Farm-Entity*. However, this concept can only be found in a few practice examples and has not yet

been scientifically studied. Therefore, a main contribution of this thesis is to describe a generalized farm architecture design for CIS and implement and evaluate that in the context of the Glossary use case. Figure 1.3 illustrates the target structure of the so called *Glossary Farm* design. The basic concept of a farm platform is to provide functions to simply create, deploy and administrate individual and independent farm entities to reduce the administration effort. Further advantages are for example (1) only one account per user and (2) link knowledge across various farm entities.

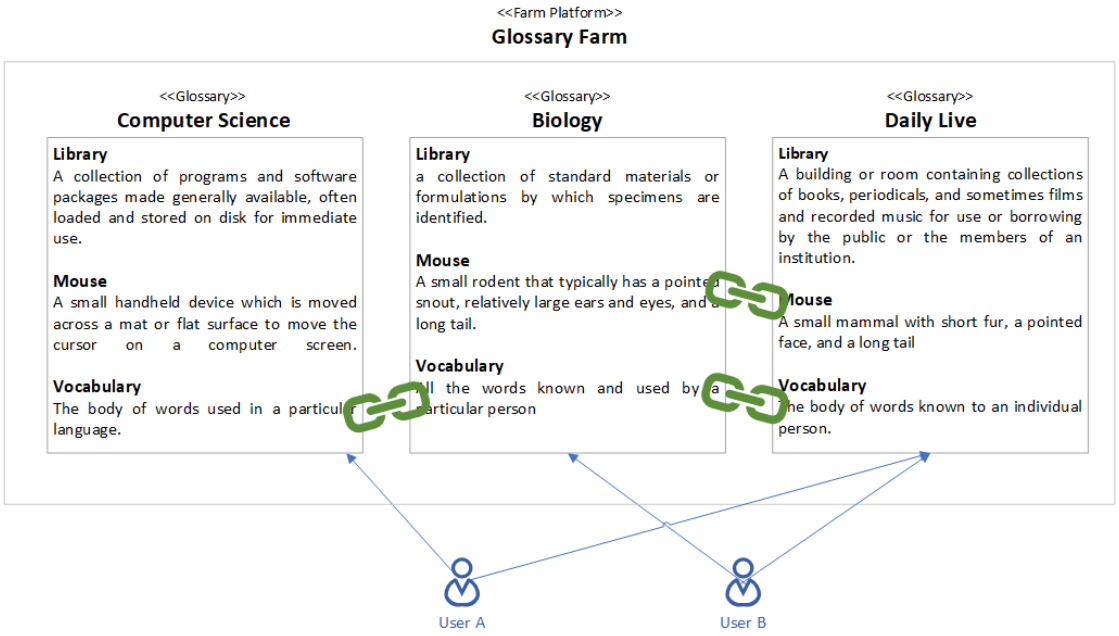

Figure 1.3: Glossary Farm Structure

Figure 1.4 presents an overview of the main contributions of this thesis. Initially a study of related work and a review of existing farm platforms are performed. Based on the review, CIS farm architectural principles are identified. This includes a specification of the system characteristics and the user needs. These results flow into the description of the CIS farm architectural approach, which is divided into the following components, (1) the *CIS Farm Meta-Model*, which illustrates the core elements and processes of the evolved architecture, (2) the *CIS-EVO-Farm Approach*, which describes a process that supports the architecture evolution of a CIS into a CIS farm platform. To verify the designed architecture approach, a prototype is implemented in the context of the Glossary. For this purpose, the requirements of a modern Glossary system are collected. The major benefit of the Glossary Farm is to ensure an efficient collection and sharing of knowledge. The designed approach is implemented in a Glossary Farm, which is finally evaluated in a user test and a survey.

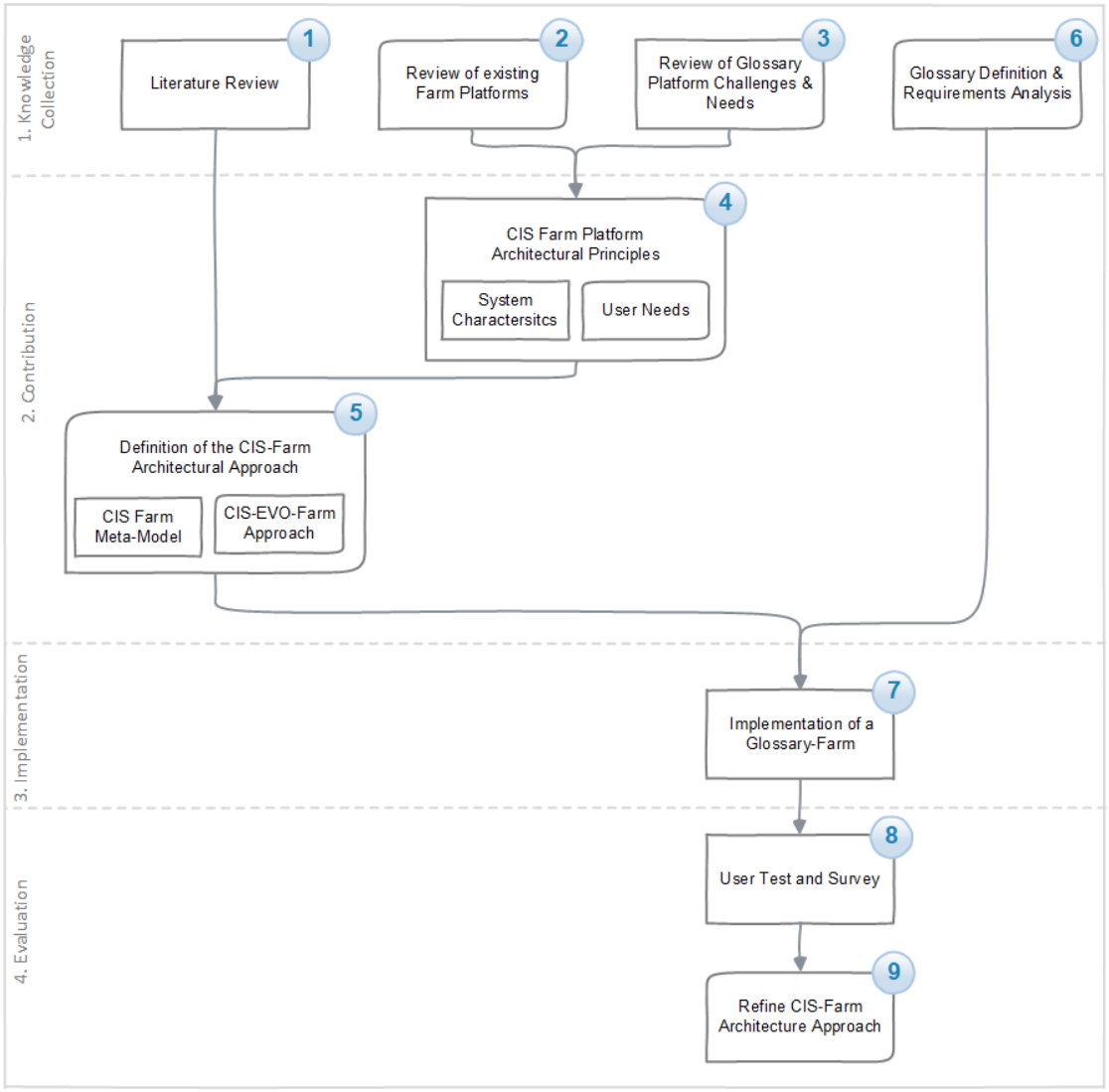

Figure 1.4: Overview of the main thesis contributions

1.4 Thesis Structure

The remainder of this thesis is structured as followed: Chapter 2 summarizes background information and related work for this thesis by starting with a general introduction of software architecture and collective intelligence systems, followed by an overview of similar architectural approaches to a farm platform. Chapter 3 defines the research questions and outlines the research methodology to answer the identified research questions. Chapter 4 focuses on the identification and comparison of existing farm platform approaches in practice. In Chapter 5 the needs and benefits of a modern glossary are presented and the Glossary Platform as a CIS is described. Chapter 6 presents the system characteristics

and benefits of a CIS farm architecture design. Chapter 7 delimits a farm to similar architectural approaches and presents the developed farm architecture with the the meta-model and describes the stigmergic process of a CIS farm. In Chapter 8 the CIS-EVO-Farm approach is introduced, which is a light-weight process to support the architecture evolution of a CIS into a farm platform. Chapter 9 describes the developed *Glossary Farm* and discuss problems during the implementation process. In Chapter 10 the results of the evaluation are summarized and analyzed. In Chapter 11 the findings of this thesis are discussed. Finally, Chapter 12 concludes this thesis and gives directions for further research work.

CHAPTER 2

Background & Related Work

This chapter summarizes background information and related work in the context of this thesis. First, a brief introduction to software architecture and collective intelligence systems is presented. Afterward, system-of-systems and multi-tenancy systems are discussed. Finally, software evolution is described.

2.1 Software Architecture

In this section we introduce software architecture and discuss its importance. Through a well-defined architecture key requirements can be fulfilled and the quality of the final product can be increased.

2.1.1 Definition

Over the last few decades, software architecture has evolved into an essential and important subfield of software engineering. A critical point in the design and construction of complex software system is the architecture. With the help of a good architecture key requirements are satisfiable, bad architecture leads to possibly fatal consequences [11].

Bass et al. [12] noticed that software architecture:

> *"[...] consists of structures and structures consist of elements and relations, it follows that an architecture comprises software elements and how the elements relate to each other."*

Another widely used definition in the literature was provided by Shaw and Garlan [13]:

> *"[...] involves the description of elements from which systems are built, interactions among those elements, patterns that guide their composition, and constrains of these patterns. [...] a particular system is defined in terms of a collection of components and interactions among those components."*

Taylor et al. [14] mentioned three fundamental methods of understanding architecture:

- Every application has an architecture.
- Every application has at least one architect.
- Architecture is not a phase of development.

The core message of the numerous definitions of software architecture states that the architecture of a system describes its gross structure. This structure highlights the top-level design decisions, including numerous things like how the system consists of interacting parts, where the main pathways of interaction are and what the main characteristics of the parts are. Software architecture can be seen as a bridge between requirements and implementation, compare figure 2.1. Through an abstract description of the system, the architecture exposes certain properties while others remain hidden [11].

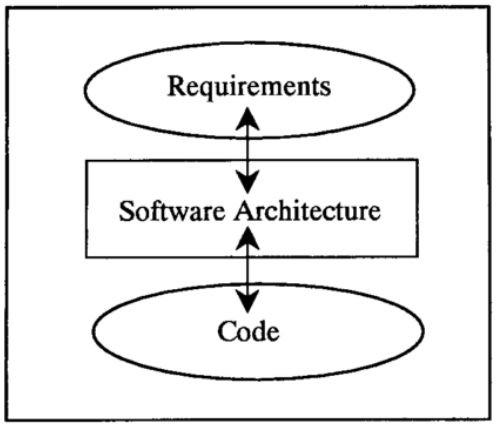

Figure 2.1: Software architecture as a bridge between requirements and implementation [11]

Software architecture plays an important role in several aspects of software development. For example, software architecture simplifies our ability to easily understand a high-level design (*Understanding*), through a partial blueprint, key components and dependencies between them are easier to identify (*Construction*) and the architectural descriptions supports the reuse at multiple levels (*Reuse*) [11].

2.1.2 Software System Life Cycle

Figure 2.2 shows the life cycle of a software system and the location of the architecture in it. As already mentioned, the software architecture links the requirements analysis with the realization. It can be incrementally researched and defined. The software architecture has to be stable before main development can be initiated. A software architecture can be considered as a blueprint and a guideline for the realization.

According to Qian et al. [15], an architecture style *"contains a set of rules, constraints, and patterns of how to structure a system into a set of elements and connectors"*. Architectural styles narrow the solution area when creating architectures. First of all, styles define which elements can exist in an architecture (like components, connectors). Secondly, they define rules for the integration of these elements into the architecture. Furthermore, styles address non-functional issues (like performance) as every style manifest its own qualities [16].

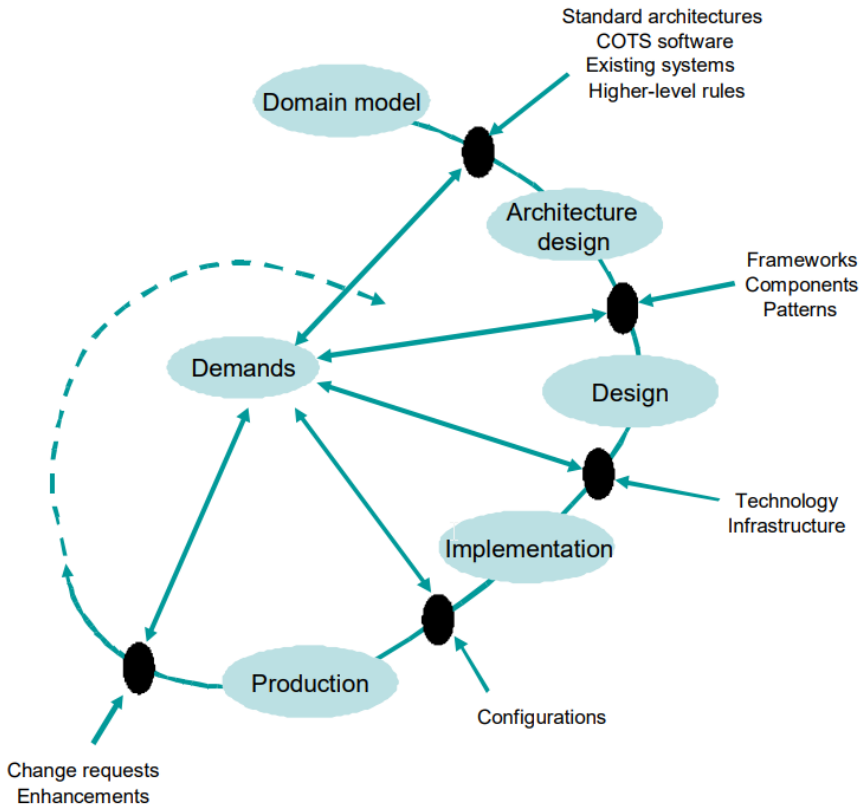

Figure 2.2: Software system life cycle [17]

The most important components of an architectural style are therefore [15]:

- *elements*, that perform functions required by a system
- *connectors*, that enable communication, coordination, and cooperation among elements
- *constraints*, that define how elements can be integrated to form the system
- *attributes*, that describe the advantages and disadvantages of the chosen structure

Clements et al. [18] assigned architecture styles to three different viewtypes. A viewtype restricts the set of elements and relations that exist within a view. First, the *module viewtype* documents the systems principal units of implementation. Secondly, the *component-and-connector (C&C)* viewtype documents the systems units of execution. Thirdly, the *allocation viewtype* maps software elements to its development and execution environments. Some common examples of architectural styles are decomposition style, layerd style, publish-subscribe style, client-sever-style, peer-to-peer style, deployment style and implementation style [18].

Each architecture has several pros, cons and potential risks. Choosing the right style to fulfill the required functions and quality features is essential. Quality attributes are identified in the requirements analysis. Quality attributes can be categorized in three main groups: (1) *implementation attributes* such as maintainability and scalability, (2) *runtime attributes* such as security, usability and availability and (3) *business attributes* like time and cost [15].

There are a variety of design principles for each software architecture, that should minimize costs and maintenance requirements and maximize usability and extendibility [19]. Five key principles are [19, 20]:

- *Separation of concerns*
  The components of the system should be split into certain functions, so that there is no overlap in functionality. Essential is minimization of interaction points, which results in a high cohesion and low coupling.
- *Single Responsibility Principle*
  Each module or component of a system should have only one specific responsibility. This should help the user to understand the system easily and it should furthermore help with the integration of the component with other components.
- *Principle of Least Knowledge*
  Each component should not know about internal details of other components. This approach avoids interdependence and helps maintainability. This principle is also known as Law of Demeter (LoD).

- *Don't repeat yourself (DRY)*
  The functionality of components should not be repeated, code should only be implemented in one component. Duplicating functionality can make it difficult to implement changes, reduce clarity and cause potential inconsistencies.

- *Minimize upfront design*
  Only design what is necessary, as not all requirements may be clearly defined or the requirements may change (agile development). This principle is sometimes known as "You ain't gonna need it" (YAGNI).

2.2 Collective Intelligence Systems

As this work seeks to extend the applicability of CIS with similar structure, we discuss hereafter the basics in detail. First we define collective intelligence and collective intelligence systems. Then we describe the characteristic CIS process, which should motivate users for further contributions. Finally, we explain an architecture approach for designing CIS.

2.2.1 Definition

Collective intelligence (CI) is one of the greatest challenges of our time. A large group of cooperating individuals produce higher-order intelligence, solutions and innovation as a single entity [21]. People used the phrase *collective intelligence* for decades, thereby a variety of definitions emerged. According to Lévy [22] CI can broadly defined as:

> *"[...] the capacity of human collectives to engage in intellectual cooperation in order to create, innovate and invent"*

Another definition is given by Malone et al. [23]:

> *"[...] groups of individuals doing things collectively that seem intelligent."*

The basic concept of CI is relative simple and exists without the use of technology. A group of individuals, for example people, insects or robots can be smarter in a way than every single member of the group. We can broadly define intelligence as *"the ability to solve problems"*. To call one system more intelligent than another system, it needs to solve more problems in the same time interval, or find better solutions for the same problems. A group has collective intelligence when it can find more or better solutions, than all the solutions found by individual working members. A large number of organizations or companies were founded with the assumption, that their members can do more together than they could do alone [24].

CI may exist without the use of technology. Nevertheless, technological means and in particular the Internet help the society to evolve their collective capabilities. This is where collective intelligence systems comes to place [21].

Collective Intelligence Systems (CIS) belong to the family of socio-technical systems and enable IT-mediated collective intelligence. As a socio-technical system, a CIS is driven by its users, who contribute content to a globally-shared virtual information space, each user benefits from the available shared information from other users [4].

At the beginning, the web was primarily one-directional, a large number of users viewed the content of a relatively small number of websites. The new approach of Web 2.0 is a bi-directional collaboration, where users are able to interact with websites and actively provide new content. In addition, other users are able to access this new content [25]. Collective intelligence is one of the key themes of Web 2.0 coined by Timothy O'Reilly [26].

The majority of Web 2.0 applications serves a large web audience, whereas CIS can be custom applications, designed for small, but highly specialized domains. Well-known examples for such systems are wikis and the online encyclopedia *Wikipedia*[1], review and rating platforms like *Yelp*[2] or question and answer platforms like *StackOverflow*[3]. The essential CIS features are similar to the design patterns of Web 2.0 applications [27]. Vergados et al. [28] took the seven principles from O'Reilly [29] and adopted them in the context of CIS. Collective intelligence applications should fulfill the following requirements (taken from Vergados et al. [28]):

1. *Task specific representations*
   Domain specific collective intelligence applications should support views of the task that are tailored to the particular domain.

2. *Data is the key*
   Collective intelligence applications are data centric and should be designed to collect and share data among users.

3. *Users add value*
   Users of collective intelligence applications know the most about the value of the information it contains. The application should provide mechanisms for them to add to, modify, or otherwise enhance the data to improve its usefulness.

4. *Facilitate data aggregation*
   The ability to aggregate data adds value. Collective intelligence applications should be designed such that data aggregation occurs naturally through regular use.

[1] `https://www.wikipedia.org`, last visited at 03.01.2019
[2] `https://www.yelp.com`, last visited at 03.01.2019
[3] `https://www.stackoverflow.com`, last visited at 03.01.2019

5. *Facilitate data access*
   The data in collective intelligence applications can have use beyond the boundaries of the application. Collective intelligence applications should offer web service interfaces and other mechanisms to facilitate the re-use of data.

6. *Facilitate access for all devices*
   The PC is no longer the only access device for internet applications. Collective intelligence applications need to be designed to integrate services across handheld devices, PCs, and internet servers.

7. *The perpetual beta*
   Collective intelligence applications are ongoing services provided to its users thus new features should be added on a regular basis based on the changing needs of the user community.

2.2.2 CIS Process

An important concept of CIS is stigmergy, which was originally introduced by Grassé [30] to describe the spatial coordination between termite societies. The term stigmergy comes from the Greek, the components of the term mean "mark" (*stigma*) and "work" (*ergon*) [24].

Heylighen [24] described the fundamental mechanism of stigmergy:

> *"[...] is that the environment is used as a shared medium for storing information so that it can be interpreted by other individuals"*

Differently to a spoken message, which is directed to a certain individual at a certain time, a stigmergic signal can be taken by any individual at any time. A spoken message, which does not reach its destination or is not understood is lost forever. In contrast, a stigmergic signal permanently stores information in a stable medium, that is accessible to anyone [24].

In Summary, stigmergy describes the indirect communication between individuals, who leave traces (knowledge) in the environment through various actions. Subsequent actions by the same or other individuals expand the previous collective knowledge, which is a benefit for every user. For example, Wikipedia is maintained by users all over the world 7 days a week and 24 hours a day by adding and modifying pages [31].

Musil et al. [4] described an essential CIS feature, the feedback loop. This represents the connection between the user base and the computational system. On a global level, hard accessible knowledge is continuously collected from situated individuals. Situatedness of individuals means the *"physical, cultural and social context"* that *"guides, constrains and partially determines intelligent activities"* [32]. On a local level, the consolidated information is distributed back to the individuals. Each user benefits from new available

information of high quality in his local space. Furthermore, each user is animated to continue the contribution of additional content into the globally-shared space. Therefore, the feedback loop forms a bridge between the local and global space [4]. For a better understanding a model of the CIS Process is graphically illustrated in figure 2.3, which includes the following steps:

1. Actors contribute content to the shared computational platform.
2. The system analyses the content data and extracts consolidated information.
3. The system distributes the information extracts among its actors.
4. Information facilitates either the actors local activity or triggers a subsequent content contribution (revisit step 1)

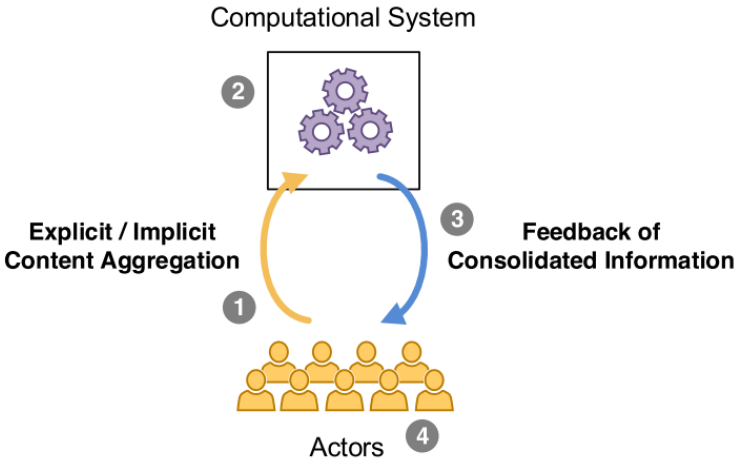

Figure 2.3: CIS process with content aggregation and feedback of information [4]

2.2.3 Stigmergic Information System Architecture Pattern

Currently, there are few research approaches to better understand and systematically design CIS. However, research on CIS is an aspiring topic and in the following, these first approaches are summarized. Musil et al. [2] defined the Stigmergic Information System (SIS) architecture pattern that described CIS architectures. This pattern addresses the problem of *"a lack of structured coordination to share and retrieve the knowledge and information between human agents by using a software system"*. Since knowledge and information are distributed among individuals, it is difficult to be aware of it and gain access to it collectively. The SIS pattern *"provides a minimal system description including the common elements and processes of a CIS"*. The SIS-based architecture design eases bottom-up information sharing and aggregation of knowledge. Firstly, actors are able to store user-generated content in topic-specific artifacts (create and modify them). Secondly, CIS enable self-organizing knowledge transfer and coordination functions for human groups and organizations.

The SIS pattern consists of the following three main layers and architectural elements [33] and is illustrated in figure 2.4:

1. The *Actor Base* is a proactive layer of *human agents* that independently interact with the system by performing *activities* on the *CI artifacts*, therefore contributing content.

2. The *Artifact Network* is a passive layer that consists of CI artifacts that store topic-specific content contributed by actors. CI artifacts are modified by *actor activities* that are similar to various types of create, read, update, and delete operations. One of the most important activities is the linking of artifacts using *artifact links*, what leads to the creation of an *artifact network*. Each performed activity is tracked in an *actor record* to assist the system in providing services such as recommendations and shared interests, whereat each actor has its own actor record. Furthermore, the actor record represents an existing *ownership relation* between the actor and the CI artifacts. The ownership relationship specify who owns an artifact, and thus has full control to decide (1) who can contribute to the CI artifact and (2) if contributions meet predefined qualitative requirements.

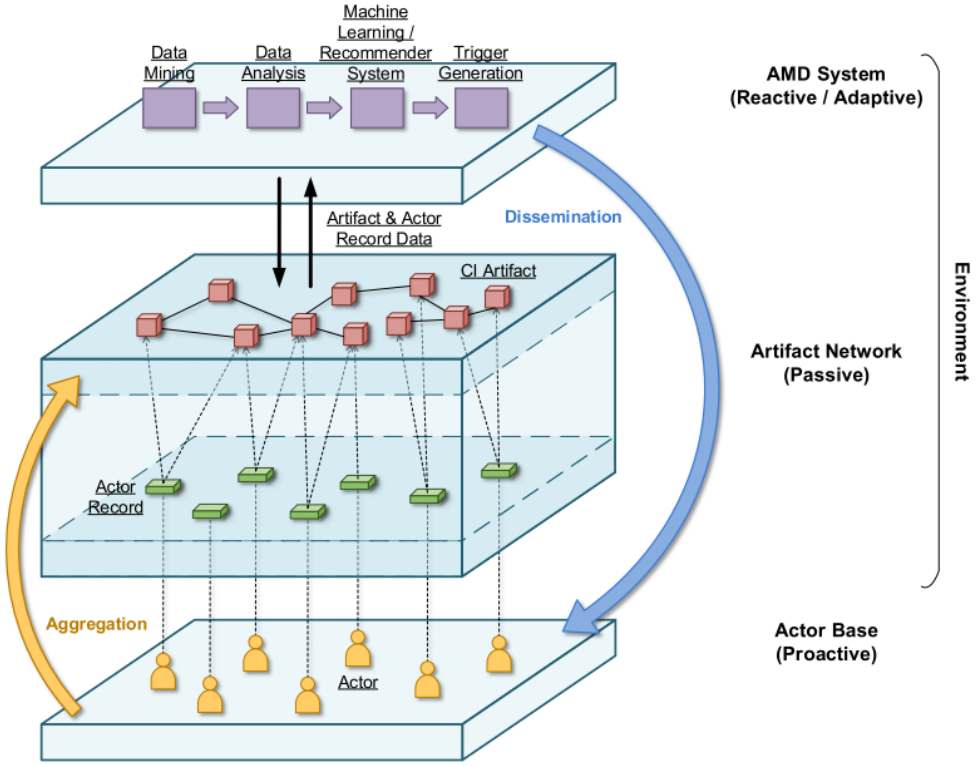

Figure 2.4: SIS multi-layer model [33]

3. The computational *Analysis, Management and Dissemination System (AMD)* represents a reactive / adaptive layer which execute different *dissemination rules*. These rules generate filtered content from CI artifact content and actor records. This filtered content is used by generated *triggers*, which are sent to individual actors to disseminate changes of CI artifacts. Among other things this should raise awareness of ongoing activities as well as stimulate subsequent actor activities and artifact contributions in order to realize the stigmergic process.

An ongoing, self-organizing coordination cycle arises through the interdependence between aggregation (collection of knowledge) and dissemination (making others aware of new or modified knowledge). This represents the stigmergic process with an aggregation and a dissemination phase [2].

Figure 2.5 shows the metamodel that underlies the SIS pattern, with the key elements and their relations. Furthermore, it emphasizes the stigmergic process including the aggregation (yellow) and dissemination (blue) phase [33].

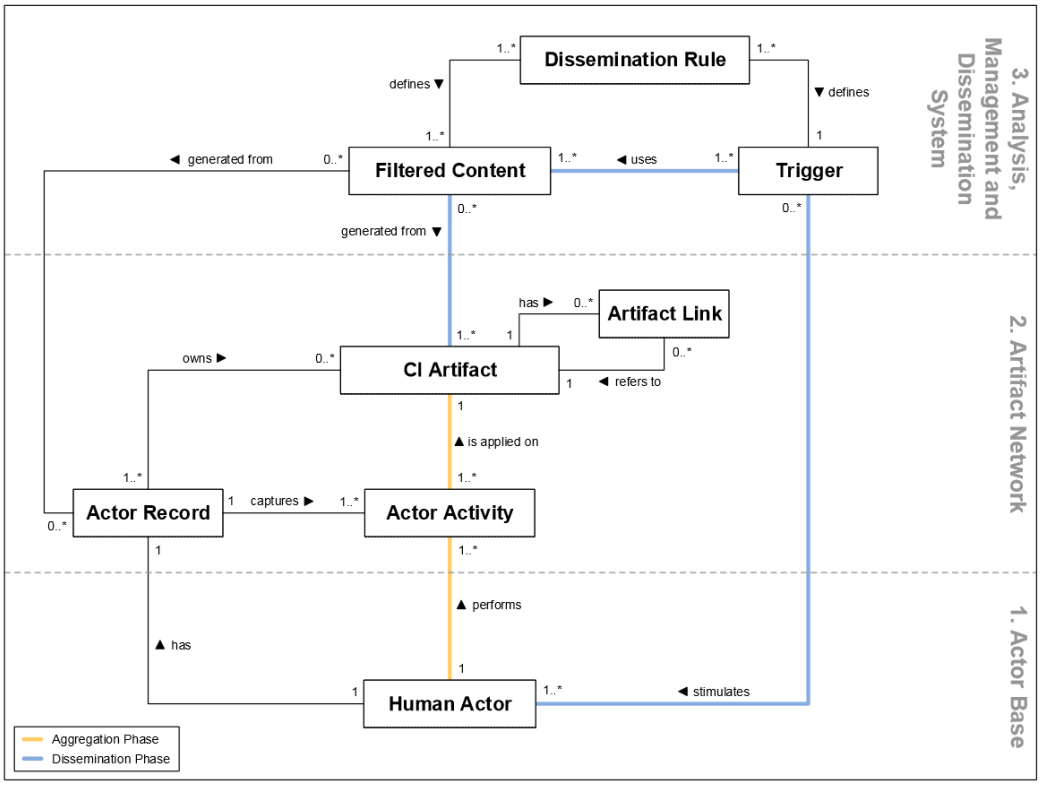

Figure 2.5: Metamodel of the SIS pattern [33]

2.2.4 Architecture Framework for Collective Intelligence Systems

Traditionally, CIS are used at society or community level. Nowadays, there is a trend to use the stigmergic mechanisms of CIS also at organization or corporate level. For example, CIS are often used in form of corporate wikis[4] to support knowledge exchange within companies. Despite the wide spread of CIS, there is a lack of architectural principles and guidance to design a CIS [31]. To address these problems, an architecture framework for collective intelligence systems (CIS-AF) was developed by Musil et al. [31]. The CIS-AF defines the key principles of CIS and provides guidance to the stakeholders for designing CIS, which are well-suited for the goals of organizations. The CIS-AF comprises three complementary viewpoints [31]:

- *context viewpoint*
  This viewpoint describes the conventions to derive an architecture view, that frames the usefulness and perpetuality concerns of architects, owners and actors that use the system.

- *technical realization viewpoint*
  This viewpoint describes the conventions to derive an architecture view that frames the data aggregation, knowledge dissemination, and interactivity concerns of architects, owners, builders, and actors.

- *operation viewpoint*
  This viewpoint describes the conventions to derive an architecture view, that frames the kickstart and monitoring concerns of system managers and analysts of CIS.

Furthermore, the CIS-AF supports the six key features of CIS [31]. The first three features include the ability of actors to *add* and *change* domain items and *create links* between data items. The remaining features comprise system support for *dissemination of selected state changes* to actors, *user-driven recommendations* and *support for tracking of usage behavior* of actors [31]. This work builds on these results.

[4] `https://www.atlassian.com/software/confluence`, last visited at 03.01.2019

2.3 System of Systems

In this section, we discuss a related architecture approach for the CIS farm architecture approach. First, we define the system of systems architecture approach, before we compare traditional software engineering with system of systems engineering. Finally, we describe how SoS can be designed.

2.3.1 Definition

In the past software systems have become increasingly large and complex, because of the interaction of several independent systems. As a result a new class of systems emerged: Systems of Systems (SoS) [34]. Silva et al. [35] defined SoS in their work as follows:

> *"[...] a widespread set of independent, heterogeneous constituent systems to form a larger system in order to accomplish a given mission, each constituent system accomplishing its own individual mission and being able to contribute to the accomplishment of the global mission of the SoS."*

SoS are formed when complex systems are joined together. This is the result when a number of operationally and managerially independent software systems are working together to fulfill a mission that no system could provide alone [8]. Each constituent system fulfills its own specific mission and furthermore contributes to the goal mission of the SoS. Through the collaboration of the constituent systems the goals of the global mission of the SoS are achieved. [35].

In the literature five characteristics for SoS are frequently mentioned (taken from Guessi et al. [8]):

- *Operational Independence*
  Constituents have autonomy to execute individually.
- *Managerial Independence*
  Constituents have their own life cycle and have autonomy for managing their own resources.
- *Emergent Behavior*
  Global functions are obtained by assembling different constituents together rather than from a single constituent.
- *Geographical Distribution*
  Constituents are not necessarily in the same environment.
- *Incremental Development*
  New functions can be added, modified, or removed in accordance with emergent needs of the SoS.

In his article Maier [36] identified three basic categories of SoS classified by managerial control:

- *Directed systems*
  These systems are built and managed to fulfill particular purposes. During long term operations it is centrally managed to continue to fulfill this purposes and any new objectives, which are defined by the system owners. Their normal operation mode is subordinated to the central managed purpose, but the component systems are able to operate independently. For instance, an air defense network is usually centrally managed to defend a region against an enemy systems, although their components may work independently.

- *Collaborative systems*
  The central management organization have no coercive power to run this system. Component systems must cooperate voluntarily to fulfill the agreed central purpose. For example, the internet is a collaborative system. The Internet Engineering Task Force elaborates standards, but has no power to enforce them. Arrangements between the central players decide how to provide or deny services, thereby they have the necessary means to enforce and maintain standards.

- *Virtual systems*
  These systems have no central management authority and, moreover, have no centrally agreed purpose. The supersystem must rely on relatively invisible mechanisms to maintain a common behavior. A virtual system arises either deliberate or accidentally. The World Wide Web is a well-known example of a virtual system, since it is distributed physically and managerially. No agency has ever centralized control, only by publishing standards for resource naming, navigation and document structure it is possible to exercise partial control. Websites decide on their own responsibility for compliance with the standards. Standards do not develop in a controlled manner, they arise from the market success of various innovators. Various forces in the World Wide Web ensure the compliance and cooperation of the core standards, they control the system in a certain way.

2.3.2 Traditional vs System of Systems Engineering

Keating et al. [37] compared in their article traditional systems engineering with System of Systems Engineering (SoSE). Traditional systems engineering has concentrated on individual complex system problems, the aim is to create one complex system for solving a specific problem or need. By contrast, SoSE centers the integration of multiple complex systems, this may involve existing systems, newly designed systems, or a hybrid mixture. The novel is the formation of the SoS as it consists of a set of integrated complex systems that originate from an emerging need or mission [37].

Further differences which were noticed by Keating et al. [37] are [38]:

- Traditional systems engineering focuses on optimization, whereas SoSE must be concerned to strive a satisfactory performance.
- Traditional systems engineering focuses on a final end product. Systems of systems evolve over time and the concept of a final solution may not be achievable. Therefore, SoSE focuses more on providing a first deployment.
- For traditional systems the requirements remain the same during development, but for SoS the requirements evolve over time.
- Traditional systems have clearly defined boundaries, whereas system systems have none.

2.3.3 Design of SoS

Ingram et al. [39] summarized several architectural principles and modeling patterns in his work for SoS and their constituent systems (CS). The five identified architectural patterns suitable for SoS design are [39]:

1. **Centralised Architecture Pattern.** A centralized architecture has a central point of control and the central CS (the *hub*) is connected to other CSs and responsible for guaranteed SoS behavior.
2. **Service Oriented Architecture.** SOA applications contain only stateless services, applications are constructed by selecting services from their service description to work together.
3. **Publish-Subscribe Architecture Pattern.** The data exchange is based on events. The *Publisher* distributes events via an event channel, registered *Subscribers* receive the published events from the event channel.
4. **Pipes and Filters Architecture Pattern.** *Filters* represent the processing steps and *Pipes* the connection between *Filters* for transferring the data. An *Input Source* represents the first step where data enters the SoS and the *Output Sink* defines where data exits.

5. **Blackboard Architecture Pattern.** The Blackboard pattern has its origin in the artificial intelligence community. Three main components work together to solve the overall problem, (1) *Knowledge Sources* solve a part of the overall problem, (2) the *Blackboard data structure* represents the central data store and (3) the *Control component* evaluates the current state of the blackboard.

Nakagawa et al. [34] did a systematic literature review about *software architecture for software-intensive SoS* including works from 2004 to 2012. In their paper it was ascertained that Service-Oriented Architecture (SOA) is an often favored architecture style for SoS. For example, the paper Simanta et al. [40] drew a parallel between SoS and SOA. Furthermore, lessons learned from SOA implementation are abstracted and applied to SoS, independent of the implementation technology.

Based on the SOA philosophy, the Arrowhead Framework was developed by Varga et al. [41]. The framework consists of systems that provide and consume services and collaborate as systems of systems. Systems that are frequently used are considered as core, such as service registration, orchestration or authorization. The Service Registry System is used to ensure that all systems can locate each other, even if the endpoints change dynamically. The Authorization system maintains a list of access rules to system resources, so that services can only be accessed by an authorized consumer. The Orchestration system is used to determine how systems should be deployed and how they can be interconnected [41].

Services are used to exchange information between a providing and consuming system (compare figure 2.6). Each service has its own set of stakeholders who are interested in the service and responsible for managing the service (definition, development, deployment and maintenance). A system can simultaneously be the service provider and service consumer of one or more services [41].

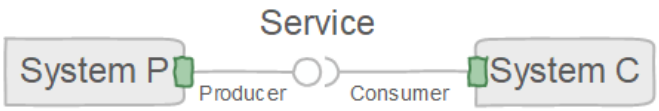

Figure 2.6: Services produced and consumed by Systems (based on [41])

In figure 2.7 the systems *A1*, *A2* and *A3* are combined together to the System of Systems *A*. This SoS *A* can be bundled together with other Arrowhead-compliant systems, such as *B1*, *C1* and *D1*. The core system *Orchestration*, *Authorization* and *Service Registry* help to create the most appropriate and secure connection between the actors. The *Dashboard Man-Machine Interface* (MMI) allows interaction with this system of system and the *Arrowhead Verification Tool* provides possibilities to test Arrowhead compliance [41].

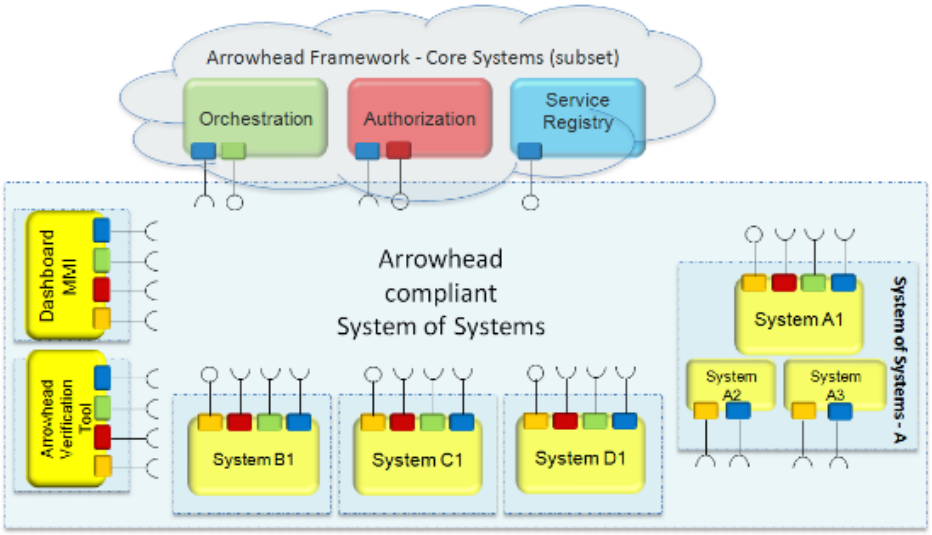

Figure 2.7: How the Arrowhead core components support the System of Systems [41]

Summary

Similar to a CIS farm, the system of systems approach also consists of several subsystems. These subsystems can band together to fulfill a mission that no system could accomplish alone. In contrast to the components of a CIS farm, these subsystems can exist alone. Nevertheless, the underlying idea is similar and we could take some architectural properties.

2.4 Multi-Tenant Systems

In this section, we discuss another related architecture approach for the CIS farm architecture approach. First, we define the multi-tenant approach and describe the stakeholders. Then we discuss the design and data storage strategies of multi-tenant systems. Finally, we describe some architecture approaches to model a multi-tenant architecture.

2.4.1 Definition

In recent years, Software as a Service (SaaS) has been an aspiring software deployment paradigm in the cloud. With this novel paradigm, companies no longer need to acquire and maintain their own ICT infrastructure. Instead, they acquire the services from a third party. The companies subscribe to the service and underlying infrastructure from a third party and require only Internet access to use these services [42]. The SaaS providers are responsible for delivering, securing and managing the application, data and underlying infrastructure [43].

SaaS is frequently offered in a multi-tenant style [9], in which a single instance of the software run on the service provider's infrastructure and several tenants access the same instance [44].

Bezemer and Zaidman [42] defined *"multi-tenant application"* in their work as:

> *"A multi-tenant application lets customers (tenants) share the same hardware resources, by offering them one shared application and database instance, while allowing them to configure the application to fit their needs as if it runs on a dedicated environment."*

In the same article Bezemer and Zaidman [42] defined the term *"tenant"* as

> *"[...] the organizational entity which rents a multi-tenant SaaS solution. Typically, a tenant groups a number of users, which are the stakeholders in the organization."*

In a multi-tenant environment, tenants share resources such as hardware and software between all users, without necessarily sharing data [9]. Nevertheless the tenants are able to configure the application to fit their individual needs. The two mainly benefits of this architecture style are the easy maintenance of the application for the service provider and the utilization of resources increases [45].

2.4.2 Stakeholders

A tenant offers the tailored application to his own group of potential users, either for in-house usage or to sell the application to customers. The individual users are unaware of the fact that other users, even from other tenants, may use the same resources and applications in parallel [46].

Different types of stakeholders can be identified in the context of multi-tenant applications [46], which is illustrated in figure 2.8:

- *Resource providers* offer the server infrastructure, including computing platforms as platform services (e.g. operating systems, databases and component containers)
- *Application providers* develop and deploy the SaaS application on top of the infrastructure
- *Tenants* rent an application
- *Users* belong to a certain tenant and access the application without the need of any installation on their own machines

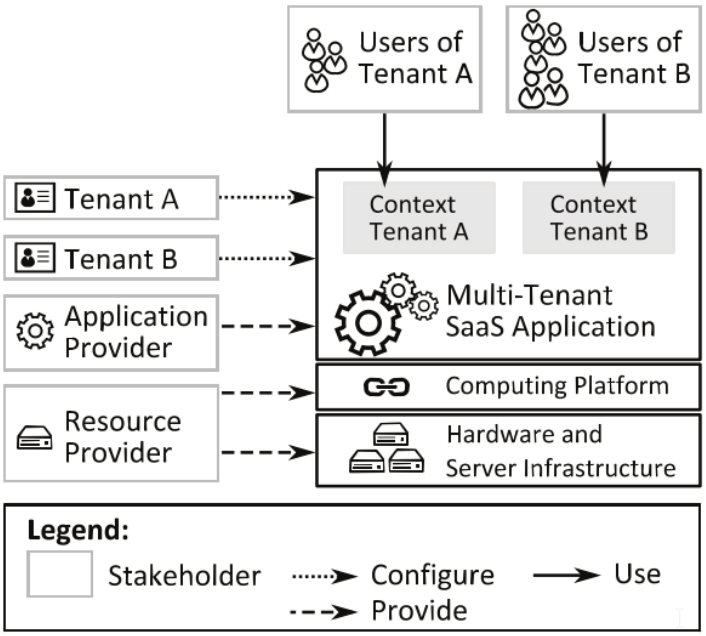

Figure 2.8: Stakeholders and their activities in a multi-tenant SaaS application [46]

Every stakeholder of a multi-tenant application has different objectives. First, users want to maximize the application qualities, e.g., select the fastest available algorithm for a computation. Different tenants want to customize the available services to their individual needs. An example of this would be to enable or disable additional services or tailor the application design to their own cooperate design. The objectives from a resource provider are to minimize the operational costs of the infrastructure. This would be possible, e.g., with using less energy or minimizing third-party license fees that are paid per used CPU. Finally, application providers are interested in maximizing the amount of tenants and their users, while minimizing the utilized resources in order to maximize their profit. This can be accomplished for example by sharing as much components as possible [46, 47].

In conclusion, it should be noted that some of these objectives represent a contradiction and have to be clarified between the individual stakeholders. For example, a contradiction is providing optimal performance, whereby reducing operation costs [46].

2.4.3 Design of Multi-Tenant Systems

There exists a difference between the concepts multi-tenant and multi-user. In a multi-user application all users use the same application with a limited possibility to configure the application. In a multi-tenant application it should be possible that each tenant has a variety of configuration options for the application. Tenants use the same building blocks in their configuration, but the design or workflow of the application could be different for two tenants [42].

Lin et al. [48] noticed, that the Service Level Agreement (SLA) of each tenant can differ, each SLA *"define a specific set of performance objective parameters"*. This is not the case for users in a multi-user application [42].

As illustrated in figure 2.9, there are two different kinds of multi-tenancy patterns. First, the multi-instance pattern, where each tenant gets his own instance of the application and possibly also the database. The native multi-tenant approach supports all tenants by a single shared application instance over various hosting resources [42, 49].

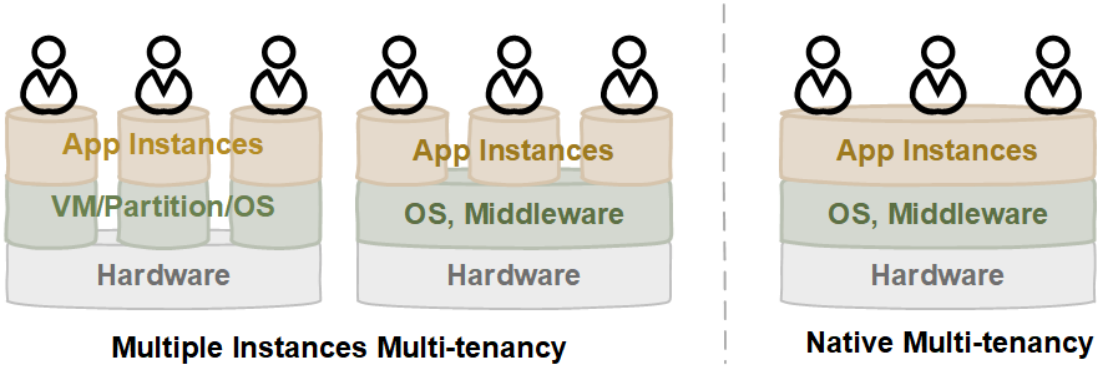

Figure 2.9: Two kinds of multi-tenancy patterns (based on [49])

The two patterns scale differently relating to the number of tenants that they are support. The multi-instance pattern is adopted to support several up to dozens of tenants. On the other hand, the native multi-tenancy pattern should support a much larger number of tenants, from hundreds to thousands of tenants [49]. In an article Guo et al. [49] noted that the *"isolation level among tenants decreases as the scalability level increases"*. When native multi-tenancy is used to support more tenants, more efforts should be made to keep the Quality of Service (QoS) at the same level for each tenant. Nevertheless, the selection of multi-tenancy technology depends on the requirements and workflow scenarios of the different target clients. For instance, a large company may prefer to pay more for multiple instances to avoid the potential risks associated with resource sharing. While most small and medium business companies would prefer services with an adequate quality at lower costs and worry less about certain types of multi-tenancy patterns, that service providers use [49].

Bezemer and Zaidman [42] claimed in their work, that the multi-instance pattern is the "easier" way of creating a multi-tenant like application from the development perspective. However, it is only better suited as long as the number of tenants remains low. With the number of tenants the maintenance costs increase, because the updates must deploy on all instances of the application [42].

2.4.4 Data Storage Strategies

Multi-tenant database designs are a widely discussed issue, i.a. by Chong et al. [50], Karatas et al. [51] and Wang et al. [52]. At the data tier there are varying degrees of data isolation for a multi-tenant application, ranging from a fully shared environment to a totally isolated environment. The three different approaches are illustrated in figure 2.10 and will be described in the following..

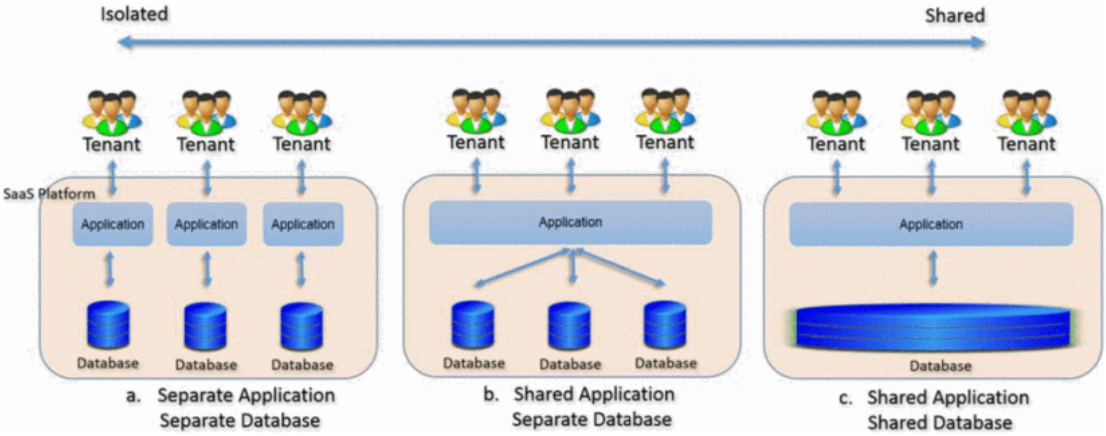

Figure 2.10: Different data storage strategies [51]

Separate Application, Separate Database - Totally isolated

Each tenant has its own separate software and database, they are totally isolated from each other [51, 52]. It is simple to extend the application's data model to fulfill individual needs for tenants, when each of them has its own database. In the case of a failure, restoring tenants data from backups is a relatively simple process. The disadvantage of this approach is that the cost for maintaining and backing up tenant data increases. Hardware costs are higher in this approach as with others, because the number of tenants that can be served on a particular database server is limited by the number of databases the server supports. Separating tenant data into individual databases is the "premium" approach, because of the comparatively high hardware and maintenance costs. This approach is suitable for customers, who are willing to pay more for additional security and customizability. For instance, customers in areas like banking or medical records management often have very strong and rigorous data isolation requirements [50].

Shared Application, Separate Database - Partially shared

The software is used by all tenants, but each user has its own physically separate database. Special methods are used, so that the software can be individually adapted to each tenant referred to his wishes [51]. The advantages and disadvantages are in most cases similar to the *Separate Application, Separate Database* approach.

Shared Application, Shared Database - Totally shared

All tenants use a common software, the approach for the database implementation is divided into two models [51], illustrated in figure 2.11.

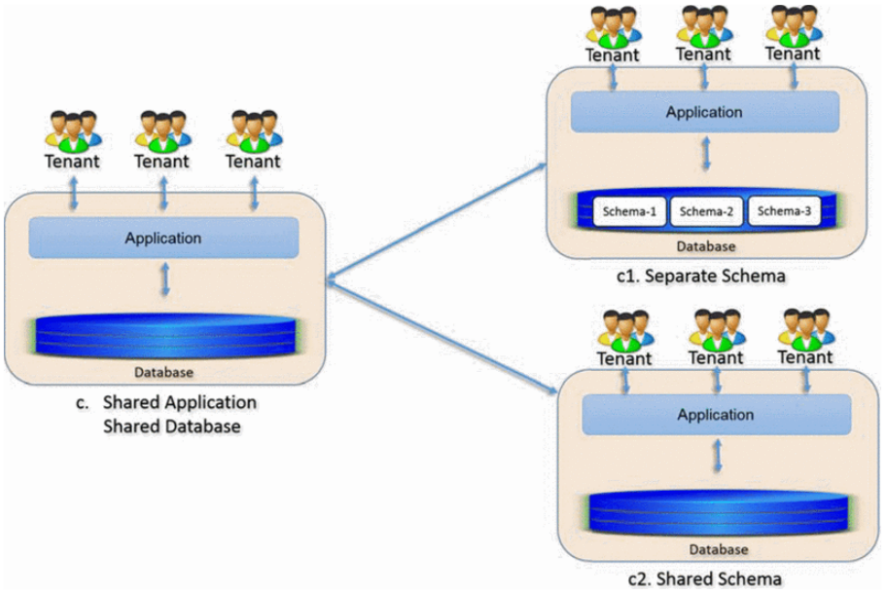

Figure 2.11: Totally shared database strategies [51]

Shared Database, Separate Schema. This approach uses a common database, but each tenant has a separate schema [51]. The separate-schema approach is relatively easy to implement, such as the isolated approach. Tenants can easily extend the data model, the tables are created only from a standard default set. However, once created, they no longer need to correlate to the default set, and tenants can add or change columns or even tables. A disadvantage of this approach is the recovery of data in case of failure. If each tenant has its own database, only one specific database needs to be restored with the data from the last backup. For a separate schema application, restoring means overwriting the data of each tenant in the same database with backup data, regardless of whether each tenant had a loss or not. The separate schema approach is suitable for applications that do not use more than 100 tables per tenant. With this approach, more clients per server can be accommodated, so the application can be offered more efficient in regard of the costs. However, customers must accept that their data are co-located with those of other clients [50].

Shared Database, Shared Schema. Multiple tenants share the same database, tables and schema [52]. The data of the tenants are only differentiated by adding a *tenantId* column [53]. This approach has the lowest hardware and backup costs, because the largest number of tenants per database server can be served. Since multiple tenants share the same database tables, this approach may cause additional development effort in the field of security to guarantee that tenants can never access other tenants data. The

effort for restoring data for a tenant is similar to the shared-schema approach with the additional complication that recovery now takes place on the row level. This approach should be used when serving a large number of clients with a small number of servers, and potential customers accept the lower data isolation in exchange for lower costs [50].

2.4.5 Architecture Approaches

Weissman and Bobrowski [54] noticed that Multi-Tenant applications are only practical when they are reliable, customizable, upgradeable, secure, and fast. To fulfill the individual expectations of various tenants a multi-tenant application must be dynamic or polymorphic. For these reasons, *Force.com* introduces a metadata architecture approach. Everything that is exposed to developers or users is internally represented as metadata. A runtime engine is used to generates application components from the metadata. Figure 2.12 illustrates the metadata-driven architecture approach. There is a clear separation between all components. These distinction makes it possible to independently update the system kernel and tenant-specific objects, with no risk to affecting others [54].

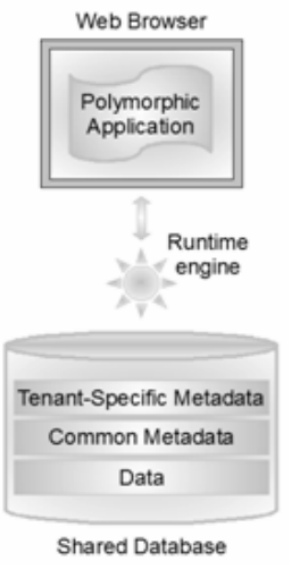

Figure 2.12: Metadata-driven architecture [54]

Jiang et al. [55] use a model-driven approach to model a SaaS application platform, which is multi-tenant aware. The model-driven approach moves the focus of software development from writing code to building models, the generation of artifacts is automated directly from the model. Figure 2.13 illustrate the architecture based on the model driven approach. The core modules of this approach are [55]:

- **Execution Platform (EP)** is mainly used for the service execution management, including the common application services and private services based on the *user id*.

According to the actions of the end users via the Internet, the appropriate services are loaded and executed.

- **Core Model-driven Engines (CME)** is the core business and controlling layer to manage the models.

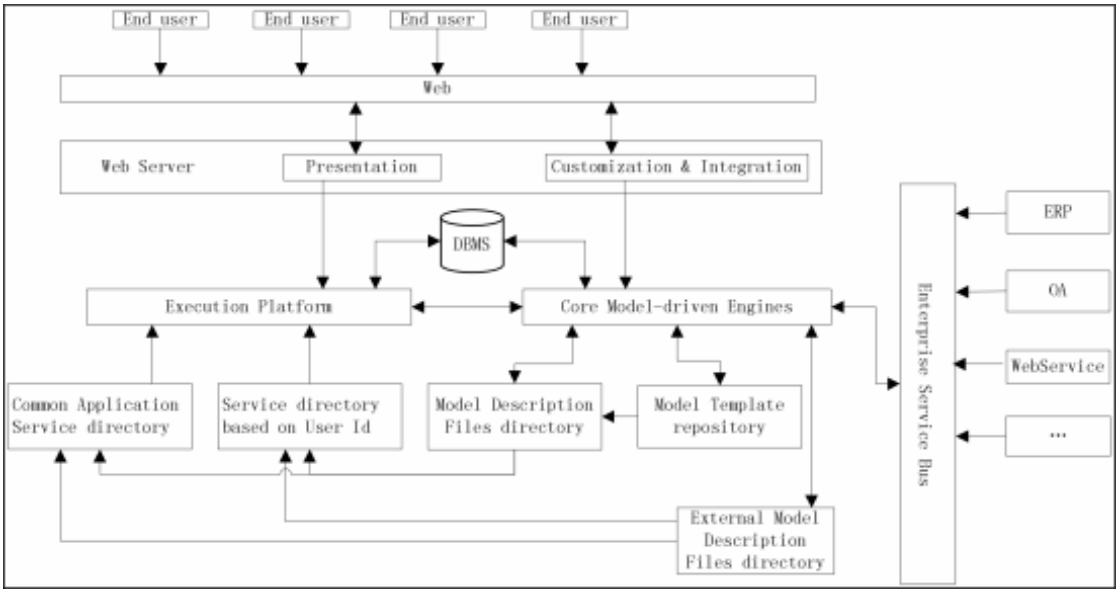

Figure 2.13: Architecture of a SaaS platform based on the model-driven approach[55]

Summary

In a multi-tenant environment, resources are shared without gaining insight into data of others. However, this is a contradiction to the aspired approach of a CIS farm. In a CIS farm, users should benefit from available shared information from other users. Nevertheless, we could take some ideas regarding the data storage strategy and architecture design.

2.5 Software Evolution

In this section, we take a closer look at software evolution since this thesis addresses mainly the concept of CIS architecture evolution.

2.5.1 Definition

A pioneer in the field of software evolution was Meir M. Lehman. In 1969 he did an empirical study within IBM with the aim of improving the company programming effectiveness. This study attracted little attention at IBM and had no impact on development practices. However, a new and prolific field of research emerged: software evolution [56].

Herraiz et al. [56] described software evolution as

> *" [...] the process by which programs are modified and adapted to their changing environment."*

The largest part of life cycle costs flows into the evolution of software to respond to changing requirements. New business opportunities occur over time, therefore there is a need to adjust the software. Because the world is constantly changing, evolvability has become a substantial quality requirement for the majority of software architectures. Commonly, the inability to evolve software systems effectively and reliably means losing business opportunities [5].

Lehman et al. [57] identified two different perspectives on software evolution: *"what"* and *"how"*. The *"what"* perspective explores the nature of software evolution and investigates the properties of this phenomenon. The *"how"* perspective focuses on essential theories, abstractions, and tools to effectively and reliably evolve a software system.

In research and practice the terms *software evolution* and *software maintenance* are often used synonymously. Nevertheless, according to Tripathy and Naik [58] there are some semantic differences between these two:

- *Software maintenance*
  This concept should prevent the software from failing to deliver the intended functionality (through bug fixing). This includes all support activities performed after the delivery of the software.

- *Software evolution*
  This concept means a continuous improvement from a smaller and simpler state to a bigger and better state. This includes all activities that are performed to effect changes in the requirements.

Based on empirical studies, Lehman and his collaborators introduced the *laws of software evolution* first in 1974. The laws continued to evolved over time, from three in 1994

to eight in 1997 (since they are unchanged). The laws are the outcome of studies to the evolution of large-scale proprietary or closed-source software (CSS) systems. These laws affect the E-type systems, a specific category of software systems [56, 58]. E-type programs are part of human or social activities. These programs became part of the modeled world (embedded in it), they try to solve problems of the involved humans [59]. The eight laws can be briefly summarized as follows [58, 60]:

1. *Continuing Change.* Systems in use must be constantly adapted, otherwise they will become increasingly less useful.
2. *Increasing Complexity.* Systems are becoming increasingly complex due to maintenance-related changes if no additional work is attempt to explicitly minimize the complexity of a system
3. *Self-Regulation.* The evolutionary process is self-regulating, the products and processes that generated during evolution are nearly normally distributed.
4. *Conservation of Organizational Stability.* The average work rate on an evolving system tends to be nearly constant over the lifetime of the system.
5. *Conservation of familiarity.* The incremental growth of systems is limited by the need to maintain a constant level of familiarity and understanding. For example, developers, salespeople and users need to understand the content and behavior of the system at all times. Too large incremental growth in one release-version reduces the understanding of the system.
6. *Continuing growth.* The functionality of systems must be continuously improved and increased to meet the needs of users and maintain their satisfaction.
7. *Declining quality.* If a system is not rigorously adapted and evolved to address changes in the operational environment, the quality will decrease.
8. *Feedback system.* Evolution processes involve multi-level, multi-loop, multi-agent feedback systems, which must also be treated as such to be continuously, successfully modified or enhanced.

2.5.2 Stage Model

Rajlich and Bennett [61] described a new way to view the software life cycle, in which they split the software lifespan in five distinct stages, each with different activities, tools, and business consequences. The five stages are illustrated in figure 2.14 and described below [61, 62]:

- *Initial development.* In this stage the first version of the software is developed from scratch. The engineers make the selection about the programming language, system architecture, used libraries and much more. These elementary choices set the course for the whole lifespan of the software. Later in the project the modification of these decisions is very expensive and in most cases not to be done.

- *Evolution.* The engineers create new features, correct former mistakes and misconception, and replay to changes in the requirements, technologies and knowledge. Software changes are the fundamental constituents of software evolution and every change introduces a new feature or property into the software.
- *Servicing.* At this stage, engineers make only minor repairs and elementary changes, since major changes are both difficult and expensive. The software should still be usable, but with minimal effort. Software in the servicing staged called ”legacy software”, ”aging software”, or ”software in maintenance”.
- *Phaseout.* The phase-out stage is entered, when the company decides that the software is not worth to do any further repairs. Nevertheless, the company tries to generate revenue from the system as long as possible - by continuing to work with it.
- *Closedown.* When the company completely retires the system from the market, it is called close-down. The company redirects users to a replacement system, if one exists.

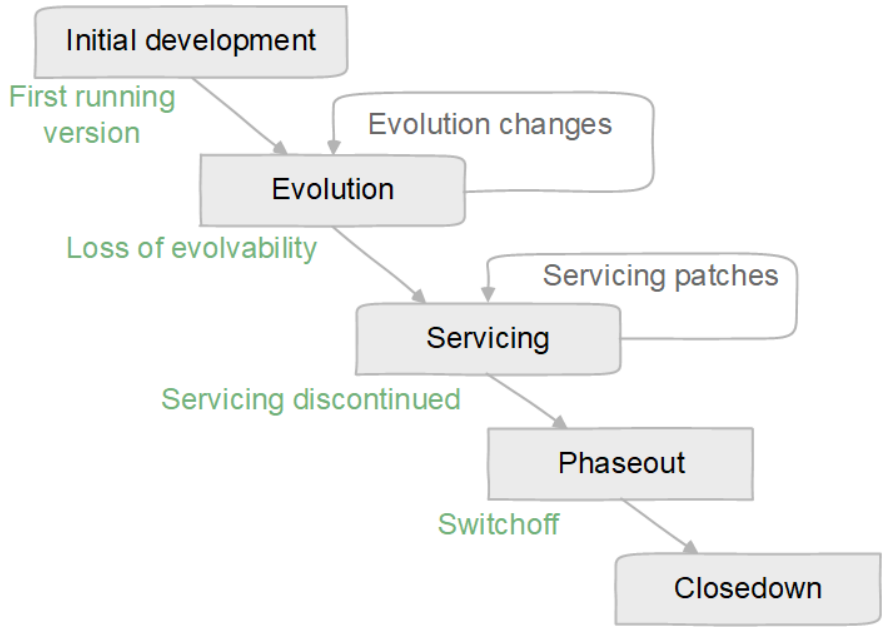

Figure 2.14: The simple staged model (based on [61])

An extension of the stage model is the versioned stage model, as shown in figure 2.15. The software team periodically creates a new version of the software, the older versions are no longer evolved, only maintained. All major changes in functionality will be implemented in future releases. If a version is outdated and users need new functionality, they will need to replace their version with a new one. Software companies that sell so-called ”shrink-wrap software” to large user communities often follow the versioned staged model [63].

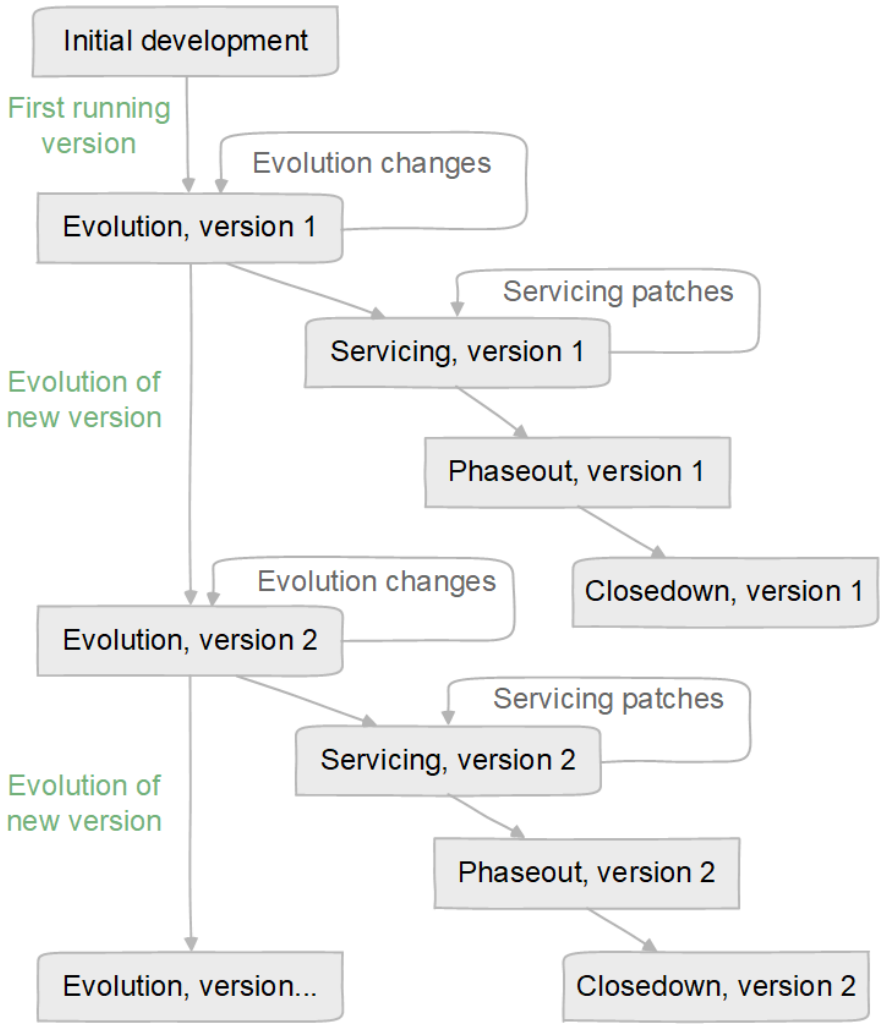

Figure 2.15: The versioned staged model (based on [61])

Software architecture can change from one stage to another. The software team specify the architecture during the initial development, which represents a significant commitment - since it determines the ease of future evolution. The architecture often loses its original clarity and integrity, because of the required changes in the system. This requires sometimes restructuring, where designers partial rebuild the architecture to facilitate future evolution. During servicing, architecture sometimes becomes an obstacle that limits the scope of possible changes. Wrapping is often applied, additional damaging the architecture by making it cryptic and difficult to understand. When code changes are performed, they should be well planned to minimize impact on other components. The deterioration lastly reaches a point where the architecture is impossible to maintain, the phase-out is the only option [61].

2.5.3 Evolution Approaches

In the following some evolution approaches and concepts from the literature are described:

- A component-based assembly of a software system leads to well-structured and modular architectures with independent and reusable components. The various components communicate with each other through their interfaces, perhaps using adapters. The component-based software system must be adjusted always to varying requirements or changes in the environment [64]. Côté et al. [64] formulates a method that supports a systematic evolution of component-based software at the architectural level. The method uses schemata and heuristics to guide software engineers through evolution tasks [64].

- Sadou et al. [65] presented in there work a model to enhance software evolution, called *Software Architecture Evolution Model (SAEV)*. Software architecture is defined by its architectural elements, such as components, connectors and configurations. To these architectural elements three different abstraction levels are associate, (1) the meta-level, (2) the architectural level, and (3) the application level. SAEV offers a whole range of concepts to describe and manage the evolution of architectures in all abstraction levels consistently. Examples for such concepts are evolution operations, evolution rules, evolution strategies and invariants Sadou et al. [65].

- The concept of online evolution addresses the issue of updating running programs without interrupting their execution. Many software systems, in particular mission critical software systems, must provide services continuously and without interruption. However, these software systems must evolve continually to fix bugs, add features, improve algorithms or adapt to new running environments [66]. Therefore, Wang et al. [66] suggest a component-based approach for online software evolution, which focus on the online update of component implementations and online update of component interfaces.

- Barnes et al. [67] outlined in there paper *" foundations for reasoning about and supporting architectural evolution"*. The focus of this approach lies on evolution paths, aiming to select an optimal path to achieve the business goals of an organization [67].

- Aoyama [68] proposes a set of metrics for software architecture evolution and discusses continuous and discontinuous software evolution on the foundation of the proposed metrics. Continuous software evolution preserves most of the aspect, whereas discontinuous software evolution changes some essential aspects, such as software architecture and major features [69]. Aoyama [68] claimed *"that discontinuity arises to reengineer software architecture and is an essential aspect of software evolution"*. The evolutionary dynamics with discontinuity exposes the inhomogeneous nature of software development over space and time.

2.6 Summary

First, a brief overview of software architecture and aspects regarding its importance were given. Through a good architecture key requirements can be fulfilled and the quality of the final product can be increased. Therefore it is necessary to find a suitable architecture to integrate several CIS instances in one environment. For the definition of an appropriate architectural approach we searched for similarities in the literature. Some similarities were identified in SoS and multi-tenant systems.

SoS arise when complex systems are joined together to fulfill a mission that no system could accomplish alone. Each system continues to pursue its own goals. However, in the intended network of CIS instances, any instance should be embedded into an enclosing system. The CIS instances need specific features that are provided by the enclosing system to work properly, e.g., at a global level it is checked whether a user has access to the CIS instance or not. Users benefit in a cross-linked environment from a network of CIS instances that link knowledge from different domains. All CIS instances have the same functionality and are only filled with different knowledge on a particular topic. However, this approach is not sufficient, because in a CIS network superordinate functions are needed.

In a multi-tenant environment resources are shared without gaining insight into data of others. However, this is a contradiction to the aspired approach of linked CIS instances, because users should benefit from available shared information of other users. Within a cross-linked environment of CIS instances users have access to several CIS instances to exchange their knowledge. Nevertheless, some ideas regarding the data storage strategy can be taken from multi-tenant systems. Summarizing, the benefits of a cross-linked environment of CIS instances are the collaboration of the users and the profit of sharing knowledge between them. These are the main achievements a multi-tenant system cannot fulfill.

The aggregation and dissemination phases are essential parts of a CIS, as their knowledge is collected and distributed. Until now, these two phases are only considered in a single, independent CIS. However, in a cross-linked environment of identical CIS instances only filled with different knowledge, makes it necessary to consider several CIS instances during the aggregation and dissemination phase.

Finally, the software evolution was considered, since most of the costs incurred in this part of the software lifecycle. Software evolution is an important issue and has not yet been processed in relation with CIS. Nevertheless, existing evolution approaches provide an orientation, e.g., the use of a component-based approach to increase the re-usability.

In conclusion, no suitable approach is currently available which addresses all challenges to embed several CIS instances into an enclosing system. Therefore it is necessary to develop a new architecture design approach.

CHAPTER 3

Research Questions and Approach

This chapter describes the research goals, the research questions and approach of this thesis. Firstly, the research challenges are specified, that occur when evolving a CIS into a more complex system. Afterwards, the research questions will be discussed and the research methods to answer them are identified.

3.1 Research Challenges

There are multiple challenges when evolving a CIS into a more complex system, where several CIS instances are integrated into an enclosing system. The existence of several similar CIS instances, only filled with different knowledge, leads to limitations, such as (1) a large number of instances to maintain, (2) many accounts for one researcher or (3) the cross-instance linking of knowledge across various instances is impossible. Therefore, a novel architecture approach needs to be developed in which multiple CIS instances can be integrated into one system environment. This creates the opportunity to remove the limitations as already mentioned. The figure 3.1 illustrates the challenges that are addressed in the context of this thesis to create a new architecture approach. The new architecture approach is illustrated in the middle of the figure and the challenges are shown on the outside. Each number in the illustration maps to the corresponding number of the list below:

1. **Missing definition**
   At the moment, there is no definition of an architecture approach to enclose several CIS instances into one environment. Therefore no architectural principles are available to design such an architecture. There are similar approaches that partly address the same problems as a farm architecture and suggest architectural principles. However, these approaches are not in the context of a CIS and do not reflect the idea of this thesis.

2. **Missing characteristics**
   There is a lack of knowledge about the system components and architecture principles of the enclosing system. Depending on the area of application or presentation of the knowledge, it may be possible to integrate different system components.

3. **Missing architecture approach**
   There are several platforms on the Internet that indicate the characteristics of the aspired architectural approach. But after a short survey, compared to each other, these platforms differ considerably and no uniform characteristics or feature set can be derived.

4. **Missing process for CIS evolution**
   There is no process available how a CIS can be evolved into a more complex system, where several CIS instances are integrated into one environment.

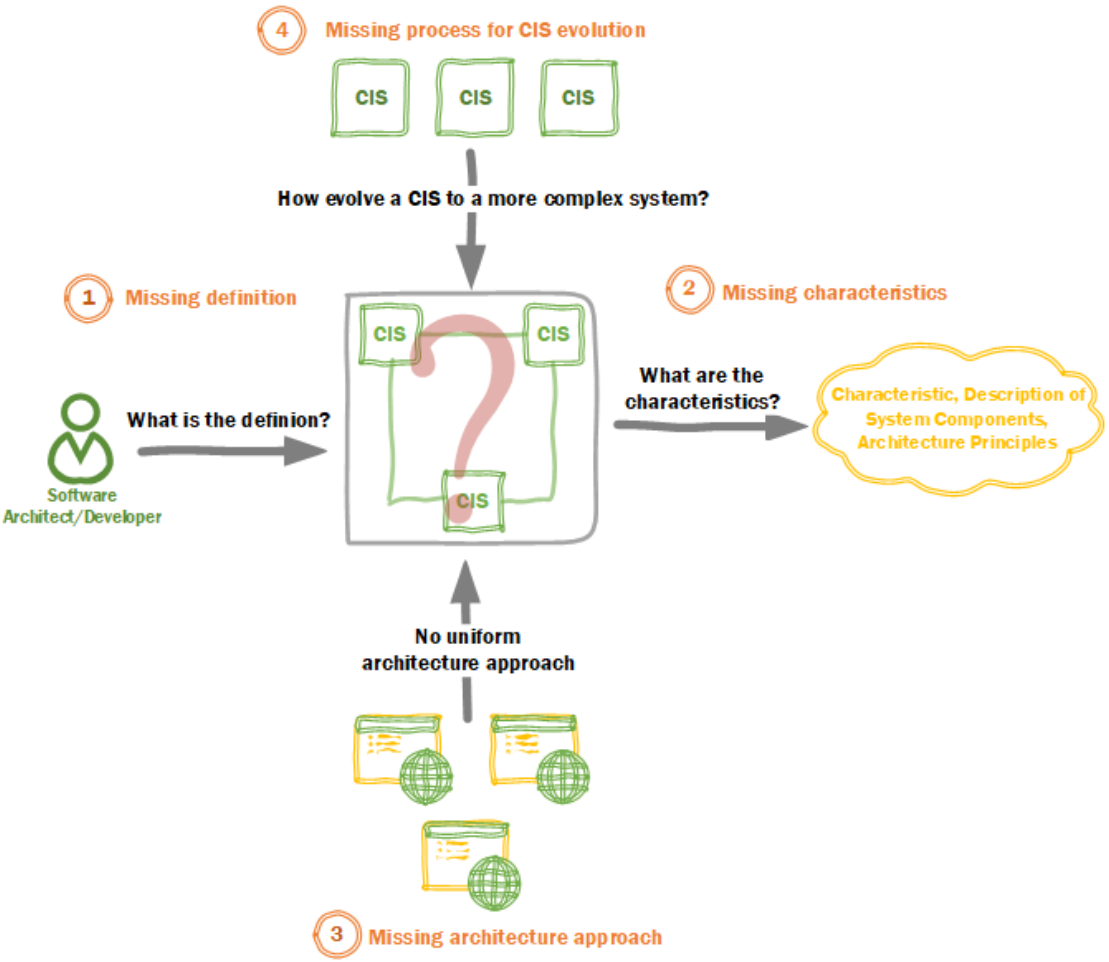

Figure 3.1: Overview of the research challenges

3.2 Research Questions

Based on these challenges, the goal of this thesis is to support software architecture evolution of collective intelligence systems (CIS) towards a so-called *CIS Farm Architecture*, which is a particular variant of a system of systems architecture. Figure 3.2 combines each research questions with the corresponding research activities and expected results. In detail, we aim to answer the following research questions to support the goal of this thesis:

RQ 1: What are the underlying architectural principles of a CIS farm platform?

To address RQ 1, we examine similar architecture approaches in literature, such as the system of systems architectures or multi-tenant systems. Additionally, systems in the field will be reviewed to identify and categorize common features and capabilities as well as variants. The consolidation of the results represents the basis for an architectural description of a CIS farm-based platform. Answering RQ 1 is important to software architects and researchers alike, since it contributes to improving the understanding of complex SoS architectures and variants in particular in the CIS application domain.

RQ 2: What are the steps to evolve a CIS into a farm platform?

This research question focuses on the challenge on how to support software architects in the evolution of CIS platforms. Therefore, we design an evolution process that should support software architects in the systematic evolution of a CIS into a CIS farm. The evolution process should consist of several steps, which are necessary to transform a CIS into a CIS farm-based platform.

RQ 3: To what extent are the identified architectural principles and the farm evolution approach sufficient to describe key characteristics of a CIS farm?

A CIS farm should enable the easy exchange of knowledge across different domains for users. The target architectural approach offers a variety of features that a single CIS instance cannot provide, e.g., one single account for several instances, link knowledge across various instances or the simple creation of a new CIS instance. The architectural principles, the CIS farm meta-model and the evolution approach should support software architects in the evolution of a CIS into a CIS farm. Therefore, we need to investigate the validity of the provided contributions of this thesis.

3.3 Research Methodology

In the following we describe the methodological approach to answer the research questions. We also illustrate the activities of the methodological approach in figure 3.2.

1. **Literature Review**
   In this step, information about existing literature on architectural principles of system of systems is collected and reviewed how it relates to the targeted CIS farm architecture approach. Another interest is to investigate the multi-tenant architectures and how they can be combined with collective intelligence systems, in particular with respect to CI-specific capabilities. Findings should provide a first insight into the farm architecture design and its characteristics as well as its application in the CIS domain.

2. **Survey of existing CIS Farm Platforms**
   In addition to a literature survey, CIS are also investigated in the field which realize such a farm architecture to identify and categorize common features and capabilities. Findings should support the identification and definition of quality indicators for CIS farms.

3. **Specification of a Glossary CIS**
   A brief analysis of glossaries will clarify their definition, benefits and requirements. Besides, current solutions to create a glossary are analyzed concerning their advantages and disadvantages. Furthermore, the needs for a Glossary CIS supporting a farm architecture should be collected.

4. **Consolidation of Architectural Principles of the CIS Farm Architecture**
   The results generated in the previous steps will provide the foundation for a design of a CIS-based SoS farm architecture. We identify the architecture principles, the characteristics and the meta-model of a CIS farm architecture approach. Through the consolidated results, we can to answer RQ 1.

5. **Definition of a CIS Farm Evolution Approach**
   This step comprises the development and definition of a CIS farm evolution approach, the CIS-EVO-Farm approach, that answers RQ 2. The CIS-EVO-Farm approach is a light-weight, decision-tree-based process that supports the architecture evolution of a CIS platform into a CIS farm. As far as applicable, existing evolution approaches should be considered as foundation to derive this approach.

6. **Architectural Design of the CIS-Farm Prototype: *The Glossary Platform***
   In this step the proposed architecture design approach and the CIS-EVO-Farm evolution approach from the previous steps are applied to evolve an existing CIS into a CIS farm. In this scenario, the existing code base and development process

information (tickets, bug reports, code reviews) of the *Glossary* Platform[1] are used. Additionally, long-time data from a specific *Glossary* CIS instance is available that was used by multi-disciplinary engineering teams from the Christian Doppler Laboratory of Software Engineering for Flexible Automation Systems, a research project of the TU Wien. Expected results of this step include an architecture description and architecture model of the to-be farm system. This step supports answering RQ 3.

7. **Development of the *Glossary Platform* prototype**
   This step focuses on the iterative development of the evolved Glossary Platform prototype following an agile software development process. For the implementation, state-of-the-art web application frameworks (Ruby on Rails) and cloud infrastructures (Heroku) will be used.

8. **User Test & Survey of Prototype Users**
   To answer RQ 3 we evaluate the Glossary Farm, where we applied our architecture principles, meta-model and evolution process. The prototype will be evaluated by conducting a user study to identify completeness, usability and usefulness of the Glossary Farm. During the evaluation we aim to answer the following questions: (1) Do the architecture model sufficiently describe the CIS farm specific elements and processes? (2) Can the CIS-EVO-Farm architecture approach satisfactorily be used to evolve a CIS to a CIS farm? (3) What success and risks factors as well as lessons learned could be identified? With the user study it should determined how well users can handle the designed concept.

9. **Refinement of the CIS farm architecture design & process based on results**
   If the results of the evaluation are not satisfactory, the CIS farm architecture design and evolution process must be revised and improved. However, if the results are satisfactory this step is skipped. A satisfactory result could be that the meta-model and the CIS-EVO-Farm approach can be used successfully during the implementation and the users gain a positive impression of the Glossary Farm during the user study.

[1] `https://glossary-cdl-sqi.herokuapp.com`, last visited at 10.06.2018

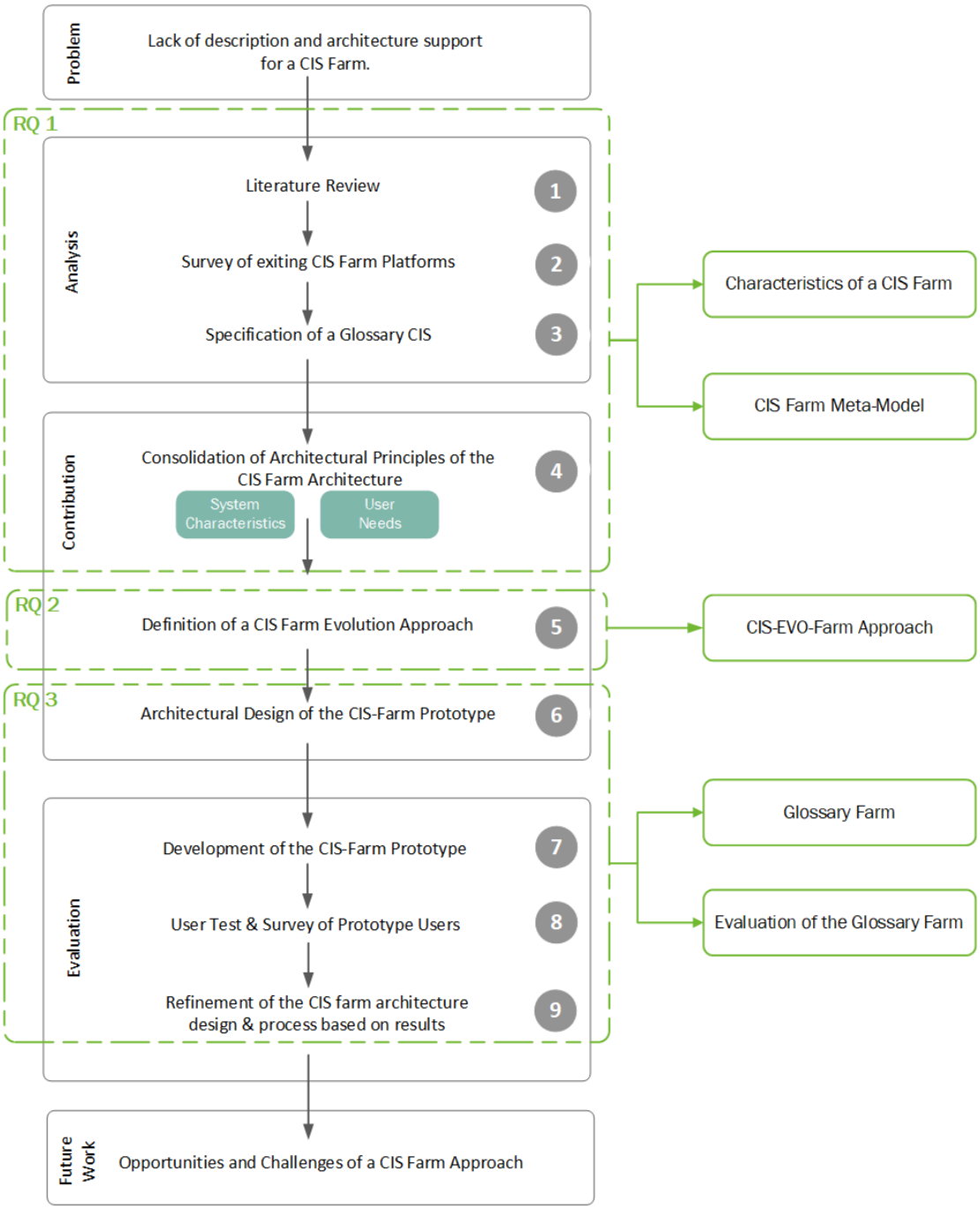

Figure 3.2: Overview of research activities in this thesis

CHAPTER 4

Survey of CIS Farm Platforms

This chapter gives an overview about existing farm approaches and comparable platforms of a farm. First, different platforms are selected for the survey, then the analysis procedure is described. Next, a brief overview of each platform is given, then they are compared in tabular form based on the selected criteria. Finally, the survey questions are answered.

4.1 Survey Design

The survey is intended to provide a first insight into the functionality of CIS farms. For this purpose, existing platforms with a farm or farm-like approach are selected and compared. Finally, common elements of the platforms are identified and described.

4.1.1 System Selection

This section describes the selection of platforms, which is divided into two parts. Firstly platforms with a farm approach are selected, then platforms with a farm-like approach.

Systems with a Farm approach

As a first step, platforms with a farm approach that differ in application and context are examined. Farm approaches for collective intelligence systems can be particularly found in wikis and Q&A sites. There are several wiki farms on the market, so a Google Trends[1] analysis was performed for the selection. Google Trends is a public website, that analyzes data based on Google search. For example, it can be shown how often a certain search term is entered in comparison to other search terms in different regions and languages. In their study, Jun et al. [70] showed that Google Trends is used to analyze various variables in areas such as IT, communications, health and economics. Google Trends makes it

[1] `https://trends.google.com`, last visited at 23.08.2018

possible to identify the current interests of searchers, allowing an immediate reflection on the needs, wishes and requirements of the users [70]. It should be noted that Google Trends is not a forecast of the future, nevertheless it can help to describe the current interests of the present [71].

Based on the Google Trends analysis (cf. figure 4.1) the two most searched providers of wiki farms for the comparison were chosen. The selected providers are Atlassian and Wikia, the former is very well-known in the business sector, the second especially in public communities.

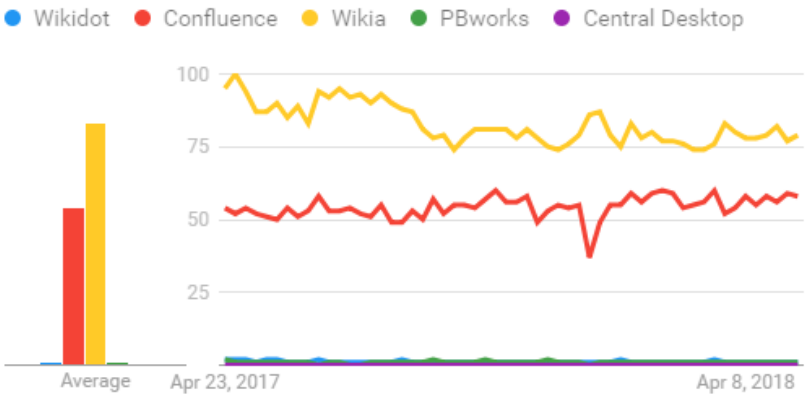

Figure 4.1: Google Trends - *Wiki-Farms*, web search between April 2017 and April 2018

In a first brief analysis about Q&A sites only Stack Exchange exhibited a farm approach, while the others were only subdivided into categories. Stack Exchange Inc. is a popular provider that hosts the well-known Q&A website StackOverflow for programmers.

Systems with a Farm-like approach

As a second step, farm-like platforms are compared to gather ideas for an architecture design. A farm-like approach is mainly used in multi-tenant computer-supported cooperative work systems. Computer-Supported Cooperative Work (CSCW) is a multidisciplinary field, in which computers are used as a key technology to support human interaction [72]. Palmer and Fields [73] defined CSCW systems as a system

> *"[...] that integrates information processing and communications activities to help individuals work together as a group."*

Multi-tenant CSCW systems share some common aspects with a farm platform. For example, multiple users use the same instance of the platform (often without their notice) to work together.

Borghoff and Schlichter [74] described in their book several CSCW application areas in detail. Incipiently with the historically first systems, such as group editors, coordination systems, message systems and conferencing systems. Networks have a major impact on how users work together to solve common tasks. Especially electronic message systems (email) and distributed file systems opened a new efficient way of cooperation, initially mainly asynchronously. Therefore, well-known representatives of CSCW systems in the field of e-mail and cloud storage are selected to get additional ideas for the design of a farm architecture.

CSCW is also a key component in customer relationship management (CRM) [72]. Since the end of 2017 CRM software has been the largest of all software markets worldwide, according to Gartner, Inc and the prognosis is, that this trend will continue [75]. Gartner, Inc. annually publishes the Magic Quadrant, which examines the global market for customer service and support applications. From the 2017 survey it can be inferred that Salesforce.com[2] is currently the leading vendor [76] and is therefore also considered in the analysis (cf. figure 4.4).

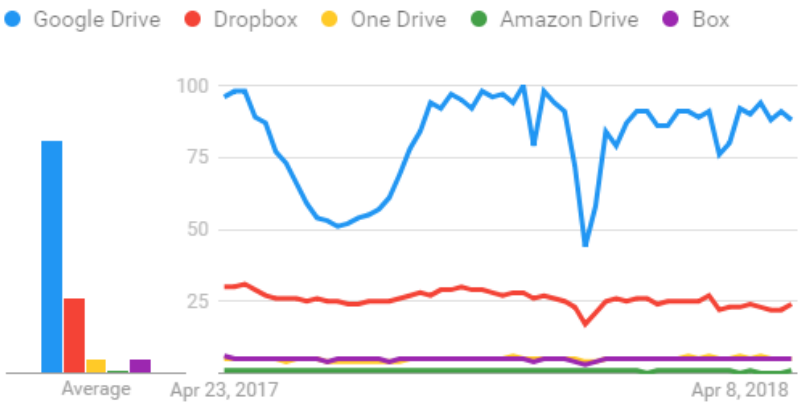

Figure 4.2: Google Trends - *Cloud Storage*, web search between April 2017 and April 2018

[2] `https://salesforce.com`, last visited at 25.08.2018

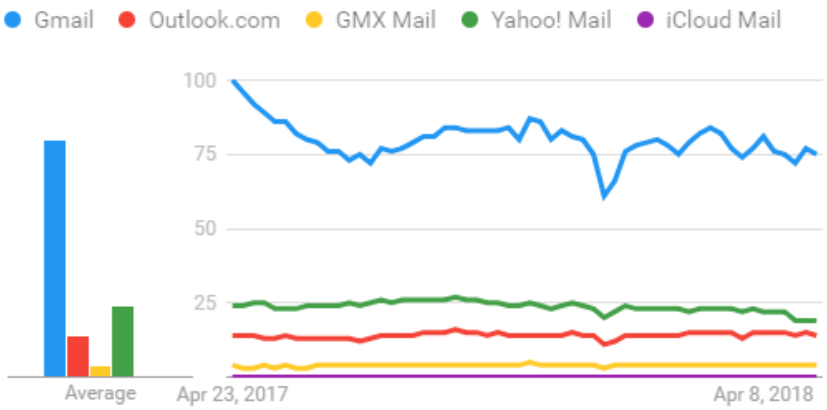

Figure 4.3: Google Trends - *E-Mail servies*, web search between April 2017 and April 2018

Figure 4.4: Magic Quadrant for the CRM Customer Engagement Center [76]

4.1.2 Survey Process and Data Collection

The analysis of each platform is based on a hands-on user experience. For each platform the same activities in an equal order are performed. However, few activity steps in the analysis differ on whether it was a platform with a farm approach or a multi-tenant CSCW system. The goal was to gain insight into how the systems work and to collect ideas for their architecture design.

A variety of data was collected about the systems during the survey, e.g. (1) how does the user management and notification system work, (2) how are the user activities recorded, (3) how can contributions be created and (4) how can a new farm entity be created. The collected data differs from system to system as they all have different fields of application. Some platforms share their knowledge in form of articles, others by asking and answering questions. Therefore, the knowledge is structured differently depending on the type of the platform. Among other things, the analysis focuses on how knowledge is available in various systems and made accessible to users.

The first steps of the data collection particularly cover the question how the aggregation of information is performed. The last steps are more focused on how the dissemination of information is performed. According to Musil et al. [2] the aggregation and dissemination phase are major topics in a CIS realization. Actors change the content of CI artifacts and the infrastructure triggers other actors through dissemination rules about the modified content. Since these phases are a main part of a CIS, particular attention is paid to reproduce them in a CIS-Farm.

Within a farm the term *farm entity* refers to an independent area in the system, e.g., inside a wiki farm the term describes a single wiki or in the Q&A farm Stack Exchange the site Stack Overflow. In a multi-tenant system, the term *farm entity* equates with tenant workspace.

For each platform with a farm approach the following activity steps were performed in the same order:

1. Create user account
2. View private profile
3. Edit private profile
4. View public profile of others
5. Create new farm entity
6. Show farm entity
7. Add item to farm entity

8. Edit item of farm entity
9. Link items over farm entities
10. Edit notification settings
11. Edit permissions of a farm entity
12. Edit own notifications

When analyzing multi-tenant CSCW systems, the following steps have been omitted 5, 9, 10 and 11. In steps 6, 7 and 8 the term farm entity refers to the tenant space.

4.1.3 Data Analysis

Each system is analyzed in a short but intensive self-study and then compared with the other systems. Through the comparison of individual platforms similarities and differences should be detected. This makes it possible to identify the system characteristics and user needs of a CIS-Farm.

During the analysis of different systems the following questions are examined and be answered:

- How is a typical farm platform structured?
- What are core components of a farm platform?
- Which similarities exist between multi-tenant CSCW-Systems and a farm?
- How are the individual farm entities separated and how can a user switch between them?
- How does the linking between different farm entities work?
- What are the characteristics of a farm?
- Which user needs can be identified regarding to a farm platform?
- How different is the community of each individual farm entity?

4.2 Survey Results

Firstly, three existing CIS-Farm approaches are presented and afterwards compared. These analyzed platforms are all based on the concept of a farm architecture, but implement that differently. Secondly, three multi-tenant CSCW systems will be presented and also compared with each other.

4.2.1 CIS-Farm Approaches

Confluence

Confluence[3] is a team collaboration software developed by Atlassian. Confluence is a commercial product, which is available as either software as a service or as on-premises software. Essentially Confluence is used as enterprise wiki for the communication and the knowledge exchange in enterprises and organizations. Spaces are used to organize content into meaningful categories. Each user can create their own space, the access to spaces can be blocked or approved for users or user groups. Through various add-ons, Confluence can be tailored to the individual needs of each team.

Fandom

Fandom powered by Wikia[4] is a free wiki hosting service, based on the open-source wiki software MediaWiki. The main purpose of Fandom is to provide information in a much larger and more comprehensive way than it is possible on Wikipedia[5]. Wikipedia has only one page per topic. In contrast, Fandom has its own wiki-instance per topic, which can be divided into various sub-pages. Each community on Fandom is specialized in a particular topic or subject. The most common interests of users on this platform are books, movies, series and games. On the contrary to other hosting services Fandom pursues the goal of founding new communities. Every hosting proposal is accepted, new wikis are automatically created within seconds. Purely private wikis are not allowed. Fandom hosts over hundred thousand wikis and is financed by advertising.

Stack Exchange

Stack Exchange is a network of question-and-answer (Q&A) websites, every single site is addressing a certain topic. The main purpose of each site is to allow users to ask questions and answer them. User are encouraged by a reputation award process to answer questions or to vote on answers and questions. Through several actions users collect reputation points, which unlock further privileges on the site. The reputation award process allows the sites to be self-moderated. Ideas for a new site must pass the Area51[6] process, this ensures that a sufficiently large community exists that can answer questions. One of the most popular sites of Stack Exchange is Stack Overflow[7] and after this initial site all other sites in the network are modeled.

[3] `https://confluence.atlassian.com`, last visited at 25.08.2018
[4] `http://fandom.wikia.com`, last visited at 25.08.2018
[5] `https://www.wikipedia.org/`
[6] `http://area51.stackexchange.com/`, last visited at 25.08.2018
[7] `https://stackoverflow.com/`, last visited at 25.08.2018

Table 4.1 shows the detailed comparison of the platforms with a farm approach.

	Confluence	**Fandom**	**Stack Exchange**
kind of platform	Wiki	Wiki	Q&A
costs	depending on the version, from 10$/month	free	free
permission to create new farm entity	everyone	everyone	Area51 process
language	various	owner choose global language	actually only English, but a Russia and a Chinese Q&A exist
single user account for all subsection	yes	yes	yes
global user profile	yes	partially	yes
farm entity user profile	no	yes	global view, but harmonized to the current entity
category	exchange knowledge about company topics	wiki in particular area: - Video games - Movies - Comics - TV - Music lifestyle	Q&A in a particular area: - Technology - Life/Arts - Culture/Recreation - Science
exchange of knowledge	user posts knowledge about a topic (internal company topics)	user posts knowledge about a topic	user asks experts for something
permissions	admin users can assign global rights, space-creator for his space	owner of the wiki can assign rights	reputation score, at the highest levels access to special moderation tools
list of all farm entities	depending on the permissions	no	yes
private farm entity	yes	no	no
navigation to other farm entities	via button	not possible	via button
linking between farm entities	possible	possible	no use
notifications	e-mail and platform notifications	e-mail and platform notifications	e-mail and platform notifications
dissemination	daily Updates recommended updates	weekly digest	best community content (weekly) question subscriptions
mobile app	yes	extra app for each wiki (not for every wiki)	yes
employees	user (company), but no special employees from Atlassian to write articles	employees write articles for the main-site	no employees to answer questions
licenses	Proprietary	Creative Commons	User contributions are licensed under Creative Commons Attribution-ShareAlike 3.0 Unported
commercial use	yes	no	no

Table 4.1: Comparison of CIS farm platforms

4.2.2 Multi-Tenant CSCW Systems

Gmail

Gmail[8] is a free, ad-supported email service developed by Google. The mail servers automatically scan emails for various purposes, such as to filter spam and malware. Gmail offers a more flexible way as other providers to organize mails, instead of folders there are *Labels* (tags) in Gmail. The mails are attached by freely defined mail filters or manually after the received to this *Labels*. Users can paste files from Google Drive into their message to send large files. Gmail offers a search-oriented interface and a conversation view, that is similar to an Internet forum. Gmail is one of the largest mail providers worldwide.

Google Drive

Google Drive[9] is a free file storage developed by Google. The service allows users to store files in the cloud, synchronize files across various devices, share files and collaboratively create and edit documents. Google Drive includes its own office suite to allow users to work collaboratively on documents. All files created through the office application are automatically saved in Google Drive. Google Drive has a sophisticated file-sharing system, the owner can control the public visibility of the file or folder. Files or folders can be privately shared with specific Google users. Sharing files with non Google account owners requires a secret URL for the file, via that the access to the file is possible.

Salesforce

Salesforce.com[10] is an international provider of enterprise cloud computing solutions. The company regards itself as a provider of Software as a Service and Platform as a Service, specializing in customer relationship management (CRM) for businesses of all sizes. The CRM service is broken down into several categories, like Service Cloud, Data Cloud and App Cloud. The products from Salesforce.com are designed to connect employees, customers and products. Through these products, organizations can manage customer accounts, track sales lines, conduct and monitor marketing campaigns, and provide after-sale service.

[8] `https://mail.google.com`, last visited at 25.08.2018
[9] `https://drive.google.com`, last visited at 25.08.2018
[10] `https://www.salesforce.com`, last visited at 25.08.2018

Table 4.2 shows the detailed comparison of the multi-tenant CSCW systems.

	Gmail	**Google Drive**	**Salesforce**
kind of application	exchange messages	store and synchronize files	CRM service: mange customer accounts, manage leads, track sales lines etc.
tenant workspace	per user	per user	per company
exchange of knowledge	only by sending e-mails, information can be exchanged	files and folders can be shared with others	only inside the company
number of users in a tenant workspace	workspace has only one user	workspace has only one user	workspace has multiple users (employees)
visibility of a tenant workspace	private, only for the owner	private, expected the shared files and folders	only per company
global user profile	none	none	only inside the company
permission	only the owner has access to his section	read or write permissions can be assigned to shared files and folders	different roles, depending on the position in the company the user has more rights (System Administrator, Contact Manager, Marketing User etc.)
structure of susection	use of labels	files are organized in folders	subdivision into apps, which have different views
list of all subsections	no	no	no
mobile app	yes	yes	yes
commercial use	no	no	yes

Table 4.2: Comparison of Multi-Tenant CSCW systems

4.3 Summary

The results of this survey are a foundation of chapter 6, where we identify the characteristics of a CIS farm architecture design. They provide an insight what are essential elements of a CIS farm architecture and are later integrated into the CIS farm meta-model.

Afterward, the analysis questions are answered based on the survey results to identify common elements:

- **How is a typical farm platform structured?**
  A farm is typically divided into areas where knowledge about a topic or domain is collected and distributed among their community. Each area is clearly demarcated and can be distinguished from the others.

- **What are core components of a farm platform?**
  Identified core components are (1) the user management, (4) the notification system, (3) the farm entity management, and (4) the artifact management in each farm platform.

- **Which similarities exist between multi-tenant CSCW systems and a farm?**
  All tenants of the investigated multi-tenant CSCW systems share the same platform instance without knowing it. One difference of this system approach, however, is that in some systems multiple users can access one tenant workspace (e.h. Salesforce). The main focus also lies on a common knowledge exchange, only realized in another way, e.g., sending messages or sharing files through separate workspaces.

- **How are the individual farm entities separated and how can a user switch between them?**
  A farm entity is a separate area in the enclosing system, which is often presented as an independent platform by hiding the other farm entities in a menu. Switching to another farm entity is usually possible through a menu or button.

- **How does the linking between different farm entities work?**
  All examined systems offer the possibility to create internal links to other farm entities. However, the creation of links is different from system to system. For example, Fandom introduces its own syntax and Confluence uses a selection dialog to create connections between farm entities.

- **What are the characteristics of a farm?**
  A farm offers an infrastructure for a variety of communities to manage, share and expand knowledge. At the same time communities can network and link related knowledge across several farm entities. Within a farm environment, each user has only one account and can be part of multiple communities.

- **Which user needs can be identified regarding to a farm platform?**
  The identified user needs are (1) single user accounts, (2) the possibility to share information between different farm entities, (3) a simple creation procedure of a new farm entity, (4) regular notifications about recent activities in the system and (5) integrated data analysis across farm entities.

- **How different is the community of each individual farm entity?**
  The community of individual farm entities can differ significantly, e.g., Stack Exchange has a community for *Anime & Manga* and *Software Engineering*.

CHAPTER 5

Application Scenario: The Glossary Platform

This chapter covers the fundamental understanding of a glossary. It is essential to know the basics of a glossary as this is used in the thesis as an application scenario. First, the term *glossary* is introduced and afterward differentiated to other similar concepts. Then the key elements and the benefits of using a glossary are presented. Furthermore, the advantages and disadvantages of available glossary solutions are discussed. Based on the available glossary solutions requirements for a modern glossary system are derived. Finally, the Glossary Platform of the TU Wien as a collective intelligence system is presented.

5.1 Definition of a Glossary

The commonality of a dictionary, encyclopedia and glossary is that they provide explanations to various terms. Nevertheless, there are significant differences between these three concepts. A dictionary is defined in the Oxford Dictionary[1] as

> *"a book or electronic resource that lists the words of a language (typically in alphabetical order) and gives their meaning, or gives the equivalent words in a different language, often also providing information about pronunciation, origin, and usage."*

[1] `https://en.oxforddictionaries.com`, last visited at 04.08.2018

A encyclopedia is defined in the Oxford Dictionary[1] as

> *"a book or set of books giving information on many subjects or on many aspects of one subject and typically arranged alphabetically."*

A glossary is defined in the Oxford Dictionary[1] as

> *"an alphabetical list of words relating to a specific subject, text, or dialect, with explanations".*

A glossary is a specialized list of words with associated definitions and can usually be found at the end of a non-fiction book. More and more, companies create standard glossaries to share words and definitions in their minds. Glossaries are commonly used for clarification and contain no information about the word like a dictionary (e.g., pronunciation). A glossary can contain both standard words and fiction words. The former can also be found in a dictionary and the second group are usually created by an organization [77]. In contrast to an encyclopedia, a glossary contains only terms with a brief definition and no comprehensive background information on a topic.

Figure 5.1 shows a typical use case of a glossary. A user checks the meaning of the word *"interpreter"* in the glossary to get a better understanding of the term. The terms and its associated definition were previously created by an expert.

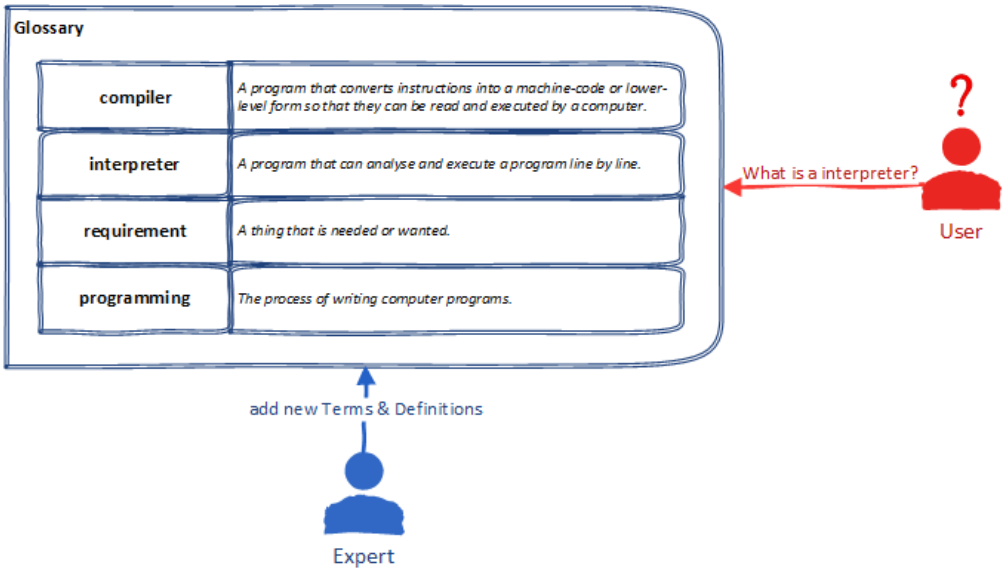

Figure 5.1: Main use case of a glossary

5.1.1 Key Elements of a Glossary

In general, a glossary is a list of terms with associated definitions and not dissimilar to a dictionary. Compared to a dictionary, a glossary contains only terms, which are unique to a business domain [78]. Key elements of a glossary include:

- **Terms.** These are unique words or short phrases that are part of the business language. Typical terms are nouns, people, places or things.
- **Definitions.** A definition indicates the clear meaning of a term in an unequivocal manner and clarifies the limits how the term can be adequately used in a communication.
- **Alias/Synonyms.** A word, phrase, or acronym that can be used exchangeable with the primary term in the glossary.
- **Related Terms.** Refer to other terms in the glossary that are similar to, but not interchangeable with, the primary term.

5.1.2 Benefits of Using a Glossary

Through the use of a glossary in a company or organization, all stakeholders have the same level of knowledge about used terms. This results in several benefits [78]:

- **Effective Communication.** A more effective communication between stakeholders is possible, as terminological disagreements can be abolished by a unified definition.
- **Common Language.** Time savings at meetings, since the participants use a common language.
- **Facilitated Learning.** Learning about a new business domain is easier, if domain-specific terms are clearly described.
- **Clear Definitions.** Misunderstandings can be avoided if important terms in documents have a clear definition.

5.2 Available Glossary Solutions

There are numerous possibilities how a glossary can be created, several are presented hereafter:

1. **Editor-based document.** An editor-based document allows the fast and simple creation of a glossary. The terms and its associated definitions can create straightforward line-by-line, through proper formatting the terms can be displayed clearly. Suitable applications are *Microsoft Word* or *OpenOffice Writer*.

2. **Table-based document.** An easy way to create a glossary is to use a table for the terms and its definitions. The first column contains the term name and the second the definition. Suitable applications are *Microsoft Excel* or *OpenOffice Calc*.

3. **Online document.** The creation procedure of a glossary in online documents is similar to editor-based documents or table-based documents. The primary advantage of this solution is that the synchronization issue is eliminated since this problem is outsourced to the vendor. A suitable applications are *Google Docs* or *Google Sheets*.

4. **LaTeX Package.** LaTeX is a document preparation system, in which the general structure of a document is defined via markup tagging conventions. The often academical used application provide a external package to create an elementary glossary.

5. **WordPress Plugin.** WordPress allows the creation and management of web pages through the browser and is nowadays also referred as a content management system (CMS). The plugin *CM Tooltip Glossary* allows the creation of an advanced glossary.

6. **Confluence Plugin.** Confluence is mainly used as an enterprise wiki for the communication and the knowledge exchange in organizations. There are plugins for the server and cloud solution available that allow the creation of a glossary.

7. **MediaWiki.** MediaWiki is free wiki software and the foundation of numerous wiki pages. A glossary can be created through an integrated function.

8. **Smartcat.** Smartcat is a translation web app that enables collaborative translation, which also allows the creation of a glossary. Certain functions of the application can only be activated by payment, such as the use of multiple account users.

There are even more applications with which a glossary can be realized, this is just a cross-section of the most well-known available solutions. In table 5.1, a few advantages and disadvantages of these solutions are listed:

Solution	**Advantages**	**Disadvantages**
Editor-based document	- Easy and fast to create	- Only textual references to synonyms and related terms - No change history - Synchronization problems between multiple users

Table-based document	- Easy and fast to create	- Only textual references to synonyms and related terms - No change history - Synchronization problems between multiple users
Online Document	- Easy and fast to create - Multi-user operation - Minimalistic change history available	- Only textual references to synonyms and related terms - Complex and confusing with many terms
LaTeX	- References to glossary entries in the document are possible - Sorting order adjustable	- Time consuming to create, due to the syntax - No change history - Text-form difficult to read (not formated)
WordPress Plugin	- Create Synonyms - Tooltips for the web page - CSV export - Multi-user operation	- WordPress application is necessary - No related terms - No history, only last editor is displayed
Confluence Plugin	- Create Synonyms, only textual (no link) - Tooltips for the web page - Term search - Multi-user operation	- Confluence application is necessary - No related terms - No history, only last editor is displayed
MediaWiki	- Change history available, but confusing and complex - Internal links to other sections possible - Clear alphabetical arrangement	- MediaWiki application is necessary - Time consuming to create, due to the syntax - No synonyms and related terms

Smartcat	- User-defined fields can be created - Term search - Multi-user operation - Export	- No history, only last editor is displayed - View is confusing with many terms - Some features are not free

Table 5.1: Advantages and disadvantages of available glossary solutions

The main purpose of the presented applications is not the creation of glossaries, so certain functions are difficult to locate or are not available. Creating terms can be cumbersome and tedious, such as learning a particular syntax. Another disadvantage is that a glossary cannot exist independently of these applications (e.g., embedded in LaTeX or Confluence). If a change in the application is being considered, because the main tasks can better be realized with another application, the formerly created glossary can sometimes not be transferred (no export or import functions available). Therefore, knowledge can sometimes be lost through the use of embedded glossaries.

Another disadvantage is that some functions are absent, such as a user-friendly and straightforward creation process for terms. Other missing features are the impossibility to view the complete term history, to create a term discussion or to vote on a definition. Furthermore, synonyms could often only be created in textual form without hyperlinks to other terms. Another problem with the first two solutions is that there are synchronization issues when multiple users are editing a glossary at the same time. It may happen, that changes get lost when two users save the document and overwrite each other. Finally, it can be said, that all presented solutions do not meet the requirements of a modern computer-based glossary system.

5.3 Requirements for a Modern Glossary System

Derived from the described problems above a wide variety of requirements of a modern glossary system could be identified. The identified requirements can be divided into essential and optional features. The basic functions of a working glossary system are illustrated in figure 5.2 and described below:

R1: Create Term. The core element of a glossary are terms, which must be created in before additional knowledge can be added.

R2: Show Term. A term must be displayed with all its stored information, such as the name of the term, creator, time of creation, etc.

R3: Terms Overview. There must be a clear representation of all terms in the system, e.g., terms sorted alphabetically.

R4: Add Definition to a Term. Each term must have at least one definition, that describes the term more precisely and specify its clear meaning in a defined context.

R5: Show Definitions of a Term. Associated definitions of a term should be displayed together with the term.

R6: Edit Term. Editing a term is a necessary feature since typos can be corrected or additional information can be added. Besides, terms can be created quickly in a discussion and completed at a later point.

R7: Delete Term. There must be a deletion function for wrong or unnecessary terms in the system.

R8: Edit Definition of a Term. A definition must be editable to complete or modify knowledge or to correct typos.

R9: Remove Definition of a Term. Definitions should be deletable to remove incorrect or duplicated definitions from a term.

R10: Add Synonyms to a Term. If a term has an interchangeable term in the system, it should be possible to hyperlink these.

R11: Add Related Terms to a Term. If a term has a related, but not identical term in the application, it should be possible to connect these.

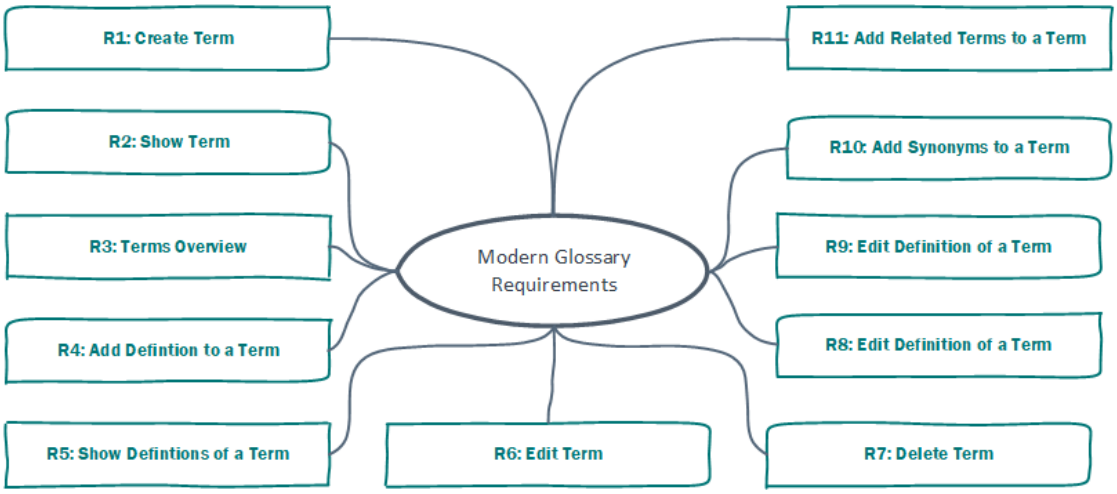

Figure 5.2: Requirements of a modern glossary

If these basic requirements are present in an application, a elementary glossary can be implemented. Additional features in the application increase the usability and the possibility to add and share knowledge more efficiently. Additional features of a modern glossary are illustrated in figure 5.3 and described below:

F1: Term History. Changes to a term and its associated definitions should be tracked. Thereby, certain changes can be undone, since information get never lost. In addition, various statistics can be generated from the activities around a term.

F2: Term Search. Numerous advantages arise when it is possible to search for a term in the application, e.g., saving time to find a term or discover related terms.

F3: Term Discussion. Users should be able to exchange knowledge on a certain term, such as incomprehensible or incomplete definitions, suggestions for additions or hints to further resources.

F4: Term Quality Level. The quality level allows the reader to quickly recognize whether a term is correct or incomplete.

F5: Definitions Quality Level. If, in addition, there is also a quality level for definitions, it is easier to identify which things still need to be revised or which definitions should be used with caution.

F6: Add External Reference to a Definition. Additional information should be added to definitions by linking to external sources, e.g., source with more detailed knowledge.

F7: Vote on a Definition. If a term has more than one definition, it may be useful to vote on the individual definitions to assert which is the best/worst definition of the term.

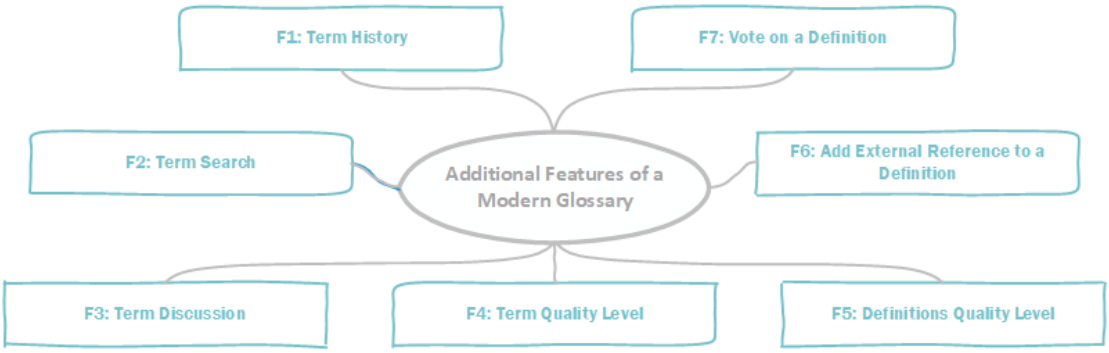

Figure 5.3: Additional features of a modern glossary

5.4 Glossary Platform as Collective Intelligence System

The Glossary Platform as a collective intelligence system is a prototypical development of the TU Wien and is already in use for research projects of the Institute of Information Systems Engineering, e.g., the *CDL-SQI Glossary*[2]. The existing platform allows scientists to collaborate geographically independently by using a shared online glossary with a collection of terms and definitions.

5.4.1 Stakeholders and their Benefits

Figure 5.4 illustrates all active stakeholders of the Glossary Platform. The *Glossary Founder* initiates the creation of a new glossary. In addition, she has to provide the necessary infrastructure to host the glossary. Maintenance of the glossary instance is the task of the *Administrator*, e.g., updates or user administration. The *Moderator* monitors all changes in terms and definitions and deletes incorrect or inappropriate contributions. *Glossary Users* adds mainly new knowledge. Besides, they change or improve existing terms and definitions. *Public Users* only view the contributions of the community.

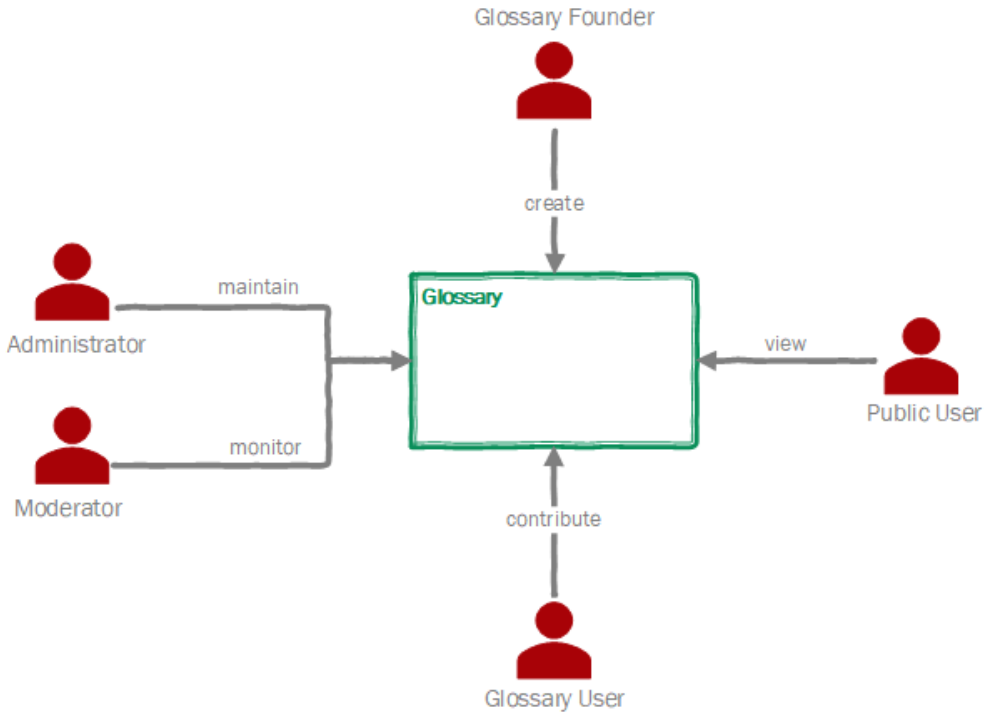

Figure 5.4: Stakeholder of the Glossary Platform

[2] `https://glossary-cdl-sqi.herokuapp.com`, last visited at 15.06.2019

The Glossary Platform allows researchers from multiple domains to collaborate geographically independently and to collect and share terms and their various definitions for a particular project or field of study. So all project participants have a common understanding of terms and concepts and reduce terminological misunderstandings and inconsistencies. New project members can also familiarize themselves more quickly, as there are uniform definitions of terms. Furthermore, the Glossary Platform aims to minimize the administration effort, which arises, e.g., from handwritten notes or Word documents, including editing, sharing, access, track keeping, monitoring, distributing, and commenting.

5.4.2 Main Use Cases

The primary use cases of the Glossary Platform can be seen in figure 5.6. Besides, the Glossary Platform implements all requirements and additional features of a modern glossary system (compare chapter 5.3).

Every glossary provides public and free access to its content, so that anyone with interest gets access. However, only registered users can contribute new information to a glossary and thereby share their knowledge with the community. Through a role model the access is restricted to certain areas, so there are some private areas that only users with specific roles can access (e.g., overview of all users and administration of them). The following roles are implemented in the glossary *administrator*, *moderator* and *standard-user*. *Moderators* differ only in one point from *standard-users*, they are allowed to delete also other contributions, not just their own.

The central concept in the *Glossary* is that different *terms* can be created and each term can possess multiple *definitions*. All logged in users can perform changes to existing term pages and add additional definitions. Since multiple users collaboratively edit a term and its definitions, all revisions of the term are logged with a list of the editors, the time-stamps and the changes that they have made. Furthermore, each term has an associated comment section, where the users are allowed to discuss about the content of the term.

Terms undergo different quality levels during their lifetime (cf. figure 5.5). New terms get the status *"not validated"*. As soon as a user thinks the term is ready and complete, she sets the status to *"needs validation"*. Subsequently, the term gets either the status *"needs rework"*, because the content is not yet matured or the status *"validated"* because everything is completed. If a term is outdated, he can get the status *"deprecated"*. Definitions go through the same quality levels and can differ from the term status.

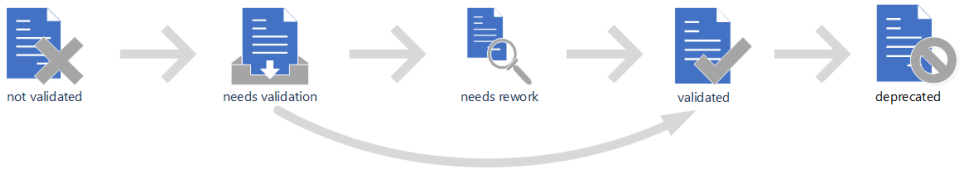

Figure 5.5: Different quality levels of terms

Another essential feature in the system are *tags*, each term and definition can be tagged with a various quantity of keywords. All terms and definitions which have been tagged with a specific keyword can be displayed on a separate overlay page. Tags are especially used to categorize content and topics, which makes it easier to discover related artifacts.

A core feature of the *Glossary* is the linking between artifacts, this is possible through *synonyms* and *related terms*. *Synonyms* are terms that mean exactly or nearly the same as another, and *related terms* are associated with others. Linked artifacts are a substantial part for the recommender system.

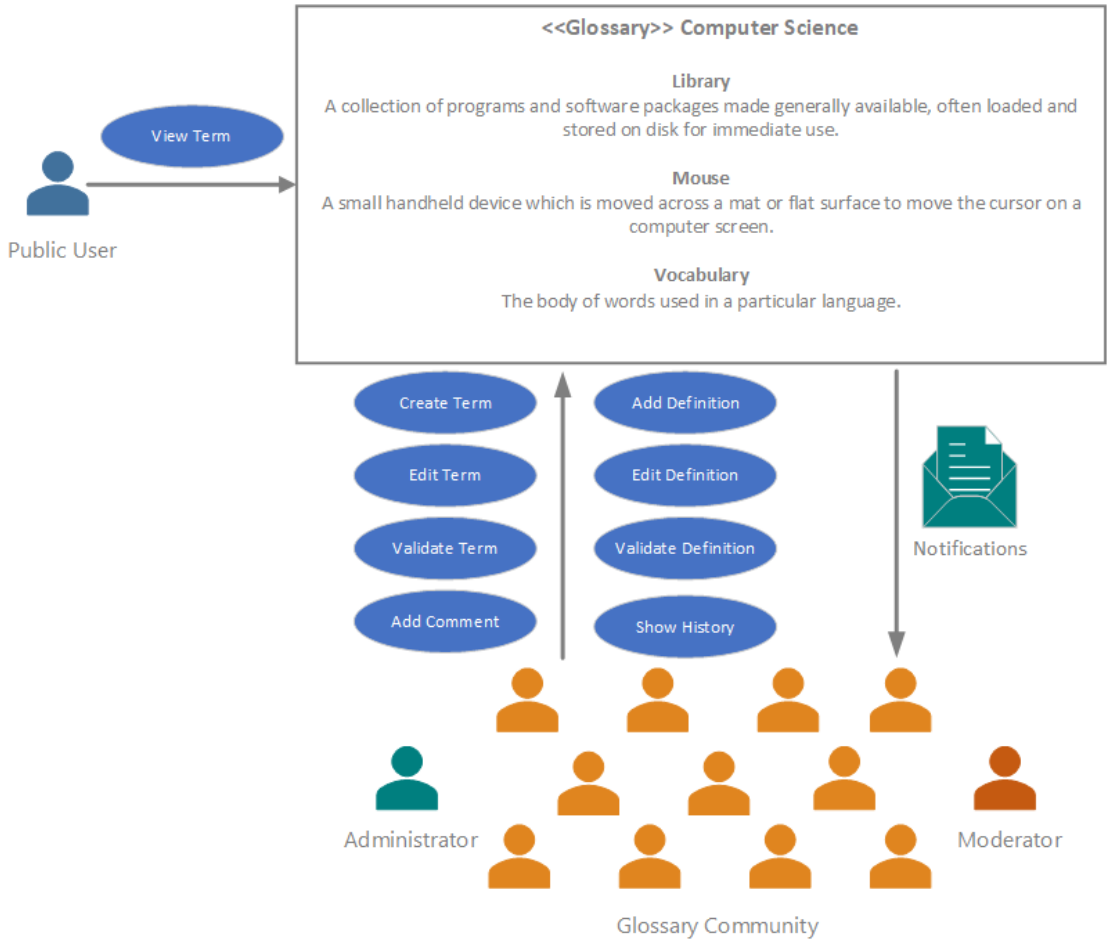

Figure 5.6: Major use cases of the Glossary Platform

All activities performed by a registered user within the *Glossary* are recorded and stored together with a reference to the account. Thereby at any time, it can be determined, for example, when a term was edited or when a user has viewed another user profile. The recorded data are summarized and published in elementary statistics inside the

glossary. Through this data, an insight into the system is possible, such as how active the community of the glossary is and how much content they generate. The main purpose of the notification system is to encourage users to further contributions. The *Glossary* uses email as the primary communication channel with the individual users. There are three different types of notifications: (1) global digest, (2) personal digest and (3) ranking summary. The global digest informs once a week about recent changes in the glossary and shows interesting stuff, such as trending terms and definitions (detected by the number of views), or artifacts, which needs validation. Weekly, the personal digest shows recent changes to own contributions and suggests similar terms and definitions for further inspection (based on calls, tags, synonyms and related terms). The ranking summary is sent monthly and provides information about the number of changes per user, the one with the most contributions gets the first place.

5.4.3 System Design & Implementation

Musil et al. [31] developed the Glossary Platform as a pilot CIS to cross-check the CIS-AF. A detailed description of the CIS-AF can be found in chapter 2.2.4. The Glossary Platform is a web application that is accessible via a web browser. The platform is implemented in Ruby on Rails and deployed on Heroku[3]. For each glossary instance, a new Heroku app must be created.

A simplified data model of the Glossary Platform is illustrated in figure 5.7. Following, a brief description of the tables and their connections:

- The most important tables are *terms* and *definitions*, since they contain the main knowledge of a glossary.
- The tables *synonyms* and *relationships* contain the connections between terms.
- All researchers who collaborates to the glossary are stored in the table *users*.
- The table *activities* contains all user activities in the system, e.g., if a user creates a new term, deletes a definition, adds a synonym to a term or tags a term.
- Table *activities* has only one direct relationship to *users*, all other connections are derived via the columns *trackable_id* and *trackable_type*.
- The table *tags* contains all tags of the glossary.
- Since several entities can be tagged, the table *taggings* has no direct connections to these. The connections are derived via the columns *taggable_id* and *taggable_type*.

[3] `https://heroku.com`, last visted on 1.12.2018

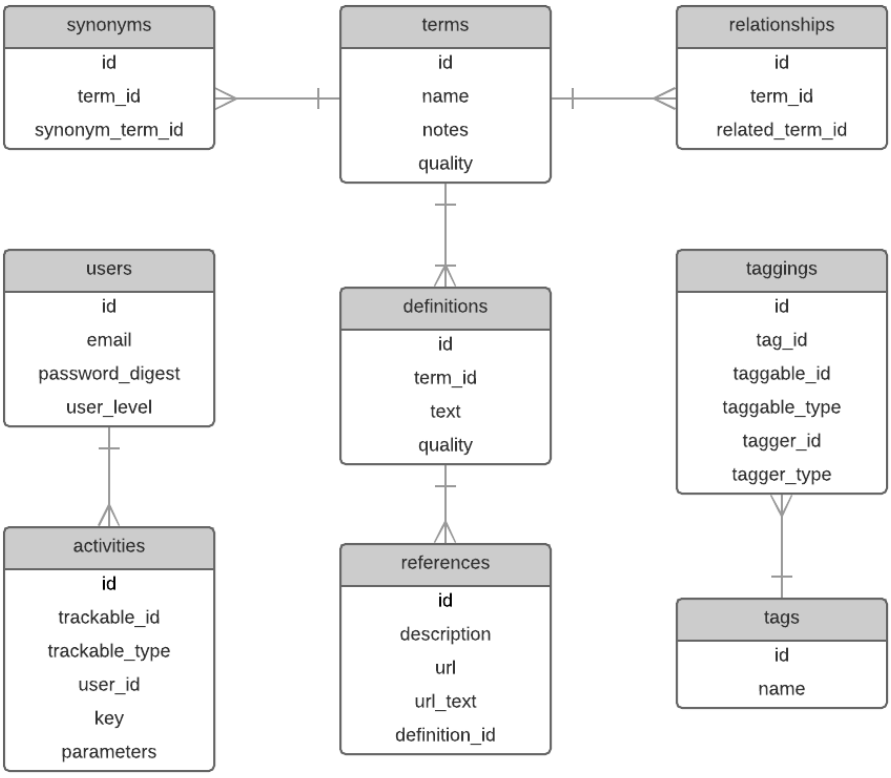

Figure 5.7: Simplified data model of the Glossary Platform

5.4.4 Challenges & Needs

Currently, a new system instance for every glossary project has to be created. This results in a significant effort for the system management and configuration by the system administrators, e.g., updates and bug fixes have to be executed separately for each system. Furthermore, a user who is involved in more than one project has to create and manage several accounts. As a result, account changes, content search across multiple projects and creating contributions in various projects becomes time-consuming and impractical.

A crucial skill that the Glossary Platform currently lacks, is a system-embracing integration. Content, its metadata and user activity data cannot be shared collectively for all glossaries. Subsequently, relevant information can be lost. For example, a new tag is added to a term in Glossary A. In Glossary B this tag is already in use. Users from Glossary B receive no notification that this tag has been linked again, although the content would be interesting for them. Users receive only demarcated stand-alone notifications per glossary.

In order to reduce the additional efforts and limitations, it is conceived to transform the

Glossary Platform into a more complex system, a so-called *Glossary Farm*. The farm approach is a particular variant of a system of systems architecture. Several glossary instances are integrated into one environment and share certain functions. With this approach, it is possible to link terms across several glossaries, create notifications based on activities from various glossaries and generate statistics and analysis over multiple glossaries.

There are many advantages of the operation of a glossary farm. Users are able to effortlessly host multiple instances of a glossary and create new communities around specific topics. The newly designed architecture also reduces the generation and maintenance expenditure for an instance. The community benefits from a single user account in various instances and the possibility of cross-reference similar information. The system-over-system software architecture allows collecting more data across the instances, thereby the opportunity for analysis increases.

CHAPTER 6

Characteristics of a CIS Farm Architecture Design

The focus of this chapter lies on the characteristics of a CIS farm architecture design. Based on preliminary investigations the system characteristics of a CIS farm are identified. Furthermore, the benefits and user needs of CIS farm are described.

6.1 System Characteristics

Based on the analysis of similar architecture concepts, the CIS farm survey of practical systems and the Glossary CIS analysis, we gain ideas for the characteristics of a CIS farm. In the following, the main characteristics of a CIS farm described. If a system fulfills all mentioned aspects, it represents a CIS farm. In addition to the textual listing, the properties are illustrated in figure 6.1.

1. *CIS farm administrator*
   A CIS farm must have an administrator, who initially sets up the system. In addition, she has to handle all major maintenance work and updates, and monitors the system.

2. *Global unique user account*
   Everybody should obtain only one account to interact with the farm and the individual farm entities. After logging in, the user can be identified in all farm entities and can perform various actions.

3. *Global individual user profile*
   Each user should obtain only one user profile in the system, to which personal data are associated. This shared, public personal data can be viewed by other users.

4. *System-wide permissions*
   Users should possess individual rights on the global level, for example the roles *administrator* and *standard-user* can be implemented. An *Administrators* can create new farm entities, invite new users or delete existing ones. *Standard-users* do not have these rights.

5. *Process to create new farm entity*
   There should be a process that allows all users to create a new farm entity. This process can be dependent on the system-wide permissions, e.g., *administrators* can create a new farm entity immediately, but for *standard-users*, the inquiry must first be approved by an *administrator*. The user who initiated the creation process becomes automatically *administrator* of this new farm entity.

6. *Classification of farm entities in categories*
   Farm entities should be subdivided into categories in order to associate them with a certain context. Due to the subdivision, the area of application is easier to identify and facilitates, e.g., the identification of relevant farm entities.

7. *Visibility of the farm entities*
   Farm entities can have either public or private visibility. Through public farm entities all users can navigate and view the artifacts. Private farm entities can only be viewed by associated logged-in users, and all others have no insight into this farm entities.

8. *User assignment to farm entities*
   Only by assigning users to farm entities they get write permissions, otherwise users have only reading rights. Through the writing permission it is possible for users to create new artifacts or to modify or accordingly supplement existing ones.

9. *Individual permission for every farm entity*
   It should be possible for users to have different rights in different farm entities. For example, a user can be *administrator* in one farm entity and a *standard-user* in all other farm entities.

10. *Artifacts belong to a farm entity*
    An artifact represents a knowledge unit and always belongs to precisely one farm entity.

11. *Create links between different farm entities*
    Similar artifacts from different farm entities can be connected within a CIS farm. This makes it possible to build an artifact network across the CIS farm.

12. *Artifact search*
    Artifacts should be searchable across all farm entities. This makes it possible to identify similar artifacts and if applicable link them.

13. *Notification system*
    The CIS farm has a global notification system to encourage users to make further contributions.

14. *Individual notification per farm entity*
    Notifications should be structured according to the farm entities, so that a lightweight distinction is possible. In addition, only messages for farm entities should be sent if an underlying action has taken place and the user is assigned to these farm entity.

15. *Notification setting for each farm entity*
    Notifications should be customizable for each farm entity. This makes it possible for users to receive all, only certain or no messages for a farm entity.

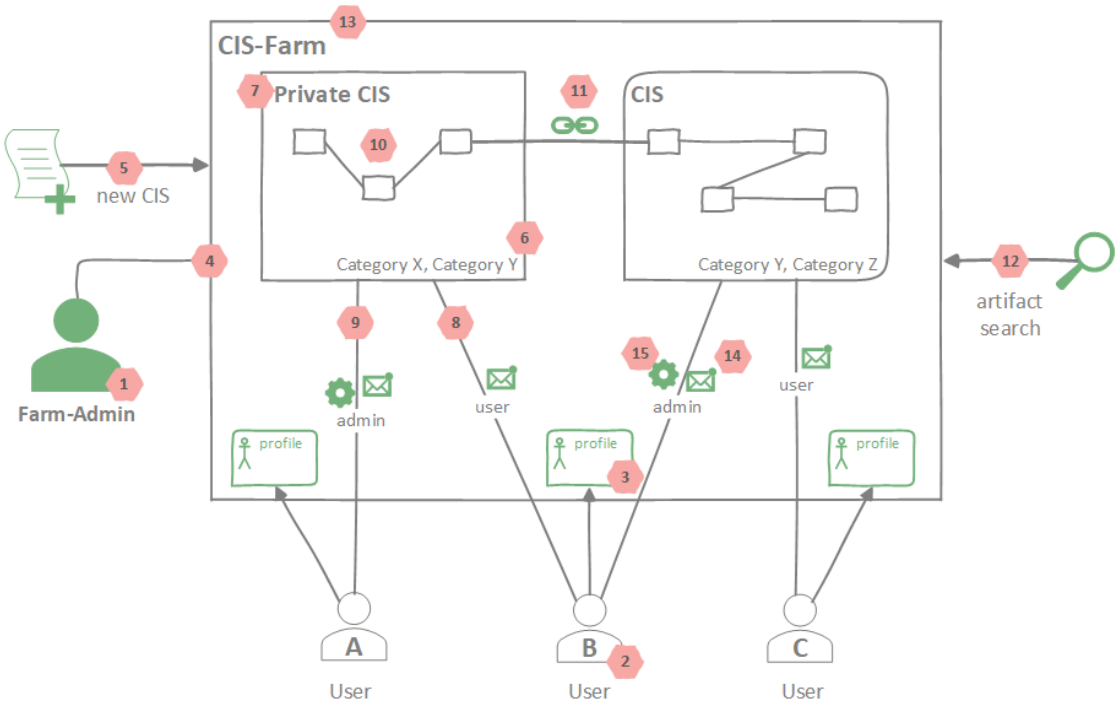

Figure 6.1: System Characteristics

6.2 Benefits of a CIS Farm Architecture

Besides the identification of the CIS farm characteristics, the investigation has also highlighted a couple of benefits and user needs. Only when all these requirements are implemented in the system, users will actively use the system and be satisfied with the application.

1. *Lower administration costs*
   The process of creating a new instance is greatly simplified, new instances need not longer be deployed and hosted separately. A farm system creates a new farm entity within the existing infrastructure. The effort to delete or temporarily deactivate an instance has been significantly reduced. In addition, there is no longer the need to manage a different URL for each instance, because the farm entities are all located within one environment.

2. *Single user account*
   Users only have a single account to interact with different instances of the platform. This allows the user to track her activities in different communities and get relevant suggestions of different instances in one notification. The effort to switch between different instances is minimized within a farm.

3. *Overview of all available farm entities*
   All instances of the farm platform can be viewed through a single action, which helps the users to find new interesting instances faster.

4. *Opportunity to share information between different farm entities*
   Inside a farm platform, artifacts from different instances can be linked expeditiously and effortlessly. Thereby it is possible to exchange information between different communities.

5. *Visibility of farm entities*
   Administrators of a farm entity have the option to mark farm entities as private, so access is only granted to explicitly invited users. This may be useful for projects that only authorized users are allowed to view.

6. *Discreet notification system*
   Users should be informed in regular intervals about activities in the system, to motivate them for further contributions. However, these notifications must not be too intrusive and, if necessary, users should be able to disable them individually for each instance.

7. *Statistics*
   Through a statistical evaluation, users have the opportunity to compare activities across different instances and may identify communities that need assistance. Beyond that, it is possible to observe the activities of the user over time and instances.

CHAPTER 7

Farm Architecture Design for CIS

At the beginning of this chapter the structure of a farm architecture will be visualized. Afterwards, a delimitation to other related architecture styles will be made. Then, the stigmergic process of a CIS farm will be described. Finally, the meta-model for a CIS farm architecture will be presented.

7.1 Terminology & Structure

Figure 7.1 shows the coarse structure of a CIS farm and commonly used terms. A *CIS farm* system consists of several *farm entities*, these entities were previously separate and independent systems.

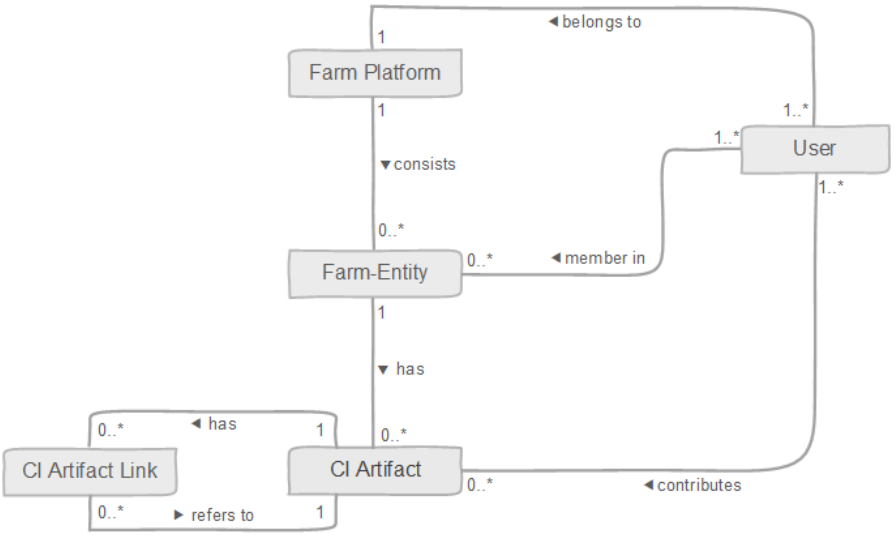

Figure 7.1: Farm Structure

In the context of a CIS farm, a CIS instance is called farm entity. Every farm entity stores its knowledge in *CI artifacts*. Each CIS farm has several *Users*, which can be member of one or more farm entities. If a user is a member of a farm entity, she is able to perform various activities on CI artifacts, such as modify an artifact or create a new one. With a *CI artifact link* it is possible to link two different artifacts together.

7.2 Delimitation to other Architecture Styles

The CIS farm architecture approach differs in several ways from already well-known architecture styles, such as the multi-tenant style or the SoS approach. These differences arise from the fact that different goals are pursued. Knowledge within a farm should be accessible to a variety of users, which should be encouraged to add new information.

7.2.1 Multi-tenancy Systems

Within a multi-tenant environment, tenants share resources, but have their own delimited area and can customize the application individually to their needs. The differences between a multi-tenant system and a CIS farm are:

1. **Tenants have only a single private area.** Each tenant gets a separate area, where its data are isolated and invisible to other tenants.
2. **Each tenant has its own users.** In several multi-tenant applications, tenants have a number of users, who have access to the data and can modify them. These users have no access to data from other tenants, therefore they can not be member of multiple tenants. Each user unambiguously belongs to a tenant.
3. **No linking of artifacts is possible over several tenants.** It is not possible to link data across different tenant spaces.
4. **Explicit release of data of a tenant.** In many multi-tenant applications, there is a mechanism to make certain data accessible to others, e.g., sharing a file in *Google Drive*. Within a public farm entity all artifacts are accessible to others.
5. **Limited dissemination.** Multi-tenant applications tend to focus on the collection of knowledge, there is no process that encourages users to further contributions. Mostly, users are not notified of changes from others.

Some types of CIS farms, such as Stack Exchange, Fandom or the Glossary Farm provide public access to all or certain farm entities, where unregistered users have read permissions. This has the advantage that more people can benefit from the knowledge in a farm entity and have the possibility to look up relevant topics. The following additional distinguishing features to a multi-tenant environment apply only to certain CIS farm types with public accessible farm entities:

- **Unregistered users have no access.** In order to gain access to the application in a multi-tenant environment, an account must first be created.

- **No global overview of all tenants.** There exists no public overview, where all tenants of a provider are listed.

7.2.2 System of Systems

Although the CIS farm approach basically consists of individual systems (also known as farm entity), several main characteristics of an SoS are violated. Among other things, the following differences between an SoS and a CIS farm could be determined:

1. **Violation of geographical distribution.** The farm entities are embedded in a single environment and can not exist alone.
2. **Impairment of operational independence.** The individual farm entities are all integrated into the same environment, they can not provide their functions independently. If, for example, the farm environment crashes due to an error, the individual farm entities can no longer be reached.
3. **No managerial independence.** Farm entities do not have an independent life cycle, they are bound to the global system.
4. **No different functionality of the individual systems.** All farm entities provide exactly the same functions, they only manage different data.
5. **Simple provision of new systems.** Since the functionalities of all farm entities are identical, there is no need to integrate new interfaces and therefore the generation of a new farm entity is a simple process.
6. **No evolutionary development.** The purpose of the global system never changes, since the farm entities always have the same functionality. It is not possible to add farm entities with other functionalities, only new features can be added to existing entities.

7.3 Architecture Overview

This section shows the stigmergic process and the meta-model of a CIS farm architecture.

7.3.1 Stigmergic Process

The stigmergic process of a CIS farm is equal to that of a CIS. The CIS farm stigmergic process consists also of two fundamental phases, which are equal to the defined process by Musil et al. [2]:

1. *Aggregation Phase.* In this phase, the actors access the content of the CI artifacts and modify them via the infrastructure.
2. *Dissemination Phase.* This phase uses active and passive dissemination rules of the infrastructure to inform actors about content changes and activities of other actors in the system.

Within a CIS farm there exist also a positive feedback loop. The stigmergic process of a CIS farm consists only of one additional element that takes care of the execution of the dissemination rules within the farm entities. Therefore, the CIS farm uses the same operating procedure as a CIS and applies an information-gathering model that is well-defined and successful. Figure 7.2 illustrates the stigmergic process in the context of a CIS farm.

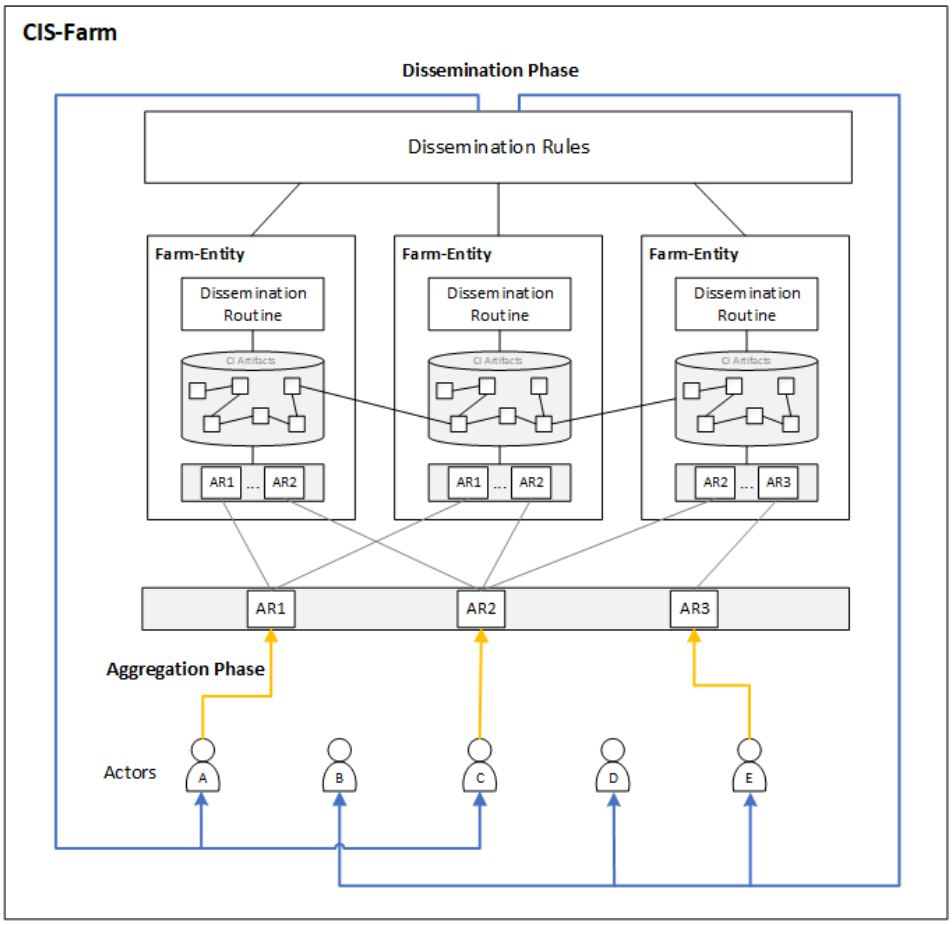

Figure 7.2: Stigmergic CIS farm process with aggregation and dissemination phases

As already mentioned, the stigmergic process implements a permanent feedback loop between a human actor base and a reactive coordination infrastructure (compare figure 7.2). A CIS farm has numerous actors, who can be members of several farm entities. The actors of a farm entity modify the content of CI artifacts, their behavior is tracked in actor records (AR). Each actor has an AR within a farm entity, that is explicitly associated with the CIS farm AR of the actor. Dissemination rules are defined for the entire CIS farm to make others aware of changes of CI artifacts. The dissemination routine in each farm entity only applies the defined dissemination rules of the CIS farm and thereby the actors are triggered to modify CI artifacts.

Example: Dissemination Phase

Dissemination rules are defined at the global level, and the dissemination routine in each farm entity deals only with the execution of this rules. Figure 7.3 illustrates the dissemination phase. The green block is executed for every member of a farm entity. The orange line is executed for all farm entities and the yellow for all members of the farm entity. The objects of the current execution cycle are always transferred to the dissemination routine (green block). After the dissemination phase is completed, all users are informed about relevant content changes and activities and may be encouraged for new contributions. Rule 1 informs users only about changes in the same farm entity, but through Rule 2 users are also informed about relevant CI artifacts from other entities.

Dissemination rules:

Rule 1: *If a user contributed to a CI artifact and another user modified the content later again, inform her.*

Rule 2: *If a user has tagged a CI artifact and another user used this tag later for a different CI artifact, inform her.*

Executed rules of the dissemination routine:

Rule 1: *Get all modified artifacts of user A in farm entity X and look if there is a later modification date.*

Rule 2: *Get all used tags of user A and look if this tag is used later for a other artifact in farm entity X from another user.*

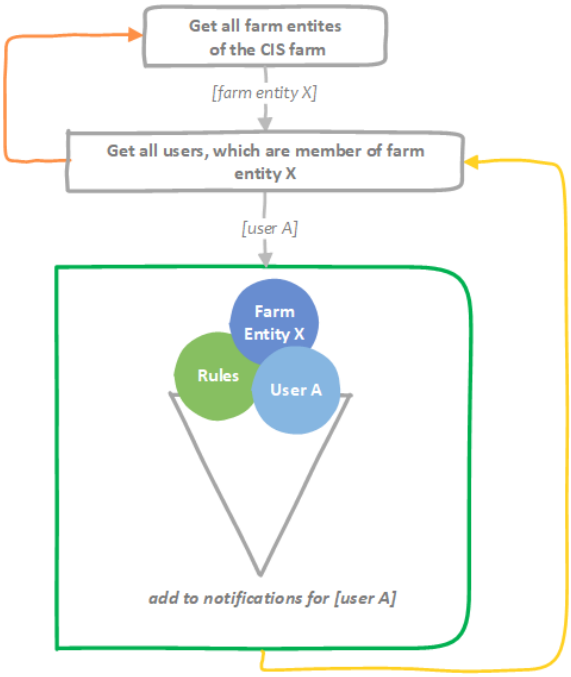

Figure 7.3: Visualization of the dissemination phase

7.3.2 Meta-Model

With the SIS architecture pattern, software architects are able to efficiently describe core elements and processes of a CIS architecture without being restricted in the technical implementation [2]. Figure 7.4 illustrates the meta-model, that underlies the SIS pattern, with the key elements and their relations. The meta-model has three main components: (1) an actor base as proactive component, (2) a CI artifact network as a passive component, and (3) an AMD system as a reactive/adaptive component [2]. For more details about the SIS pattern see *chapter 2.2.3*.

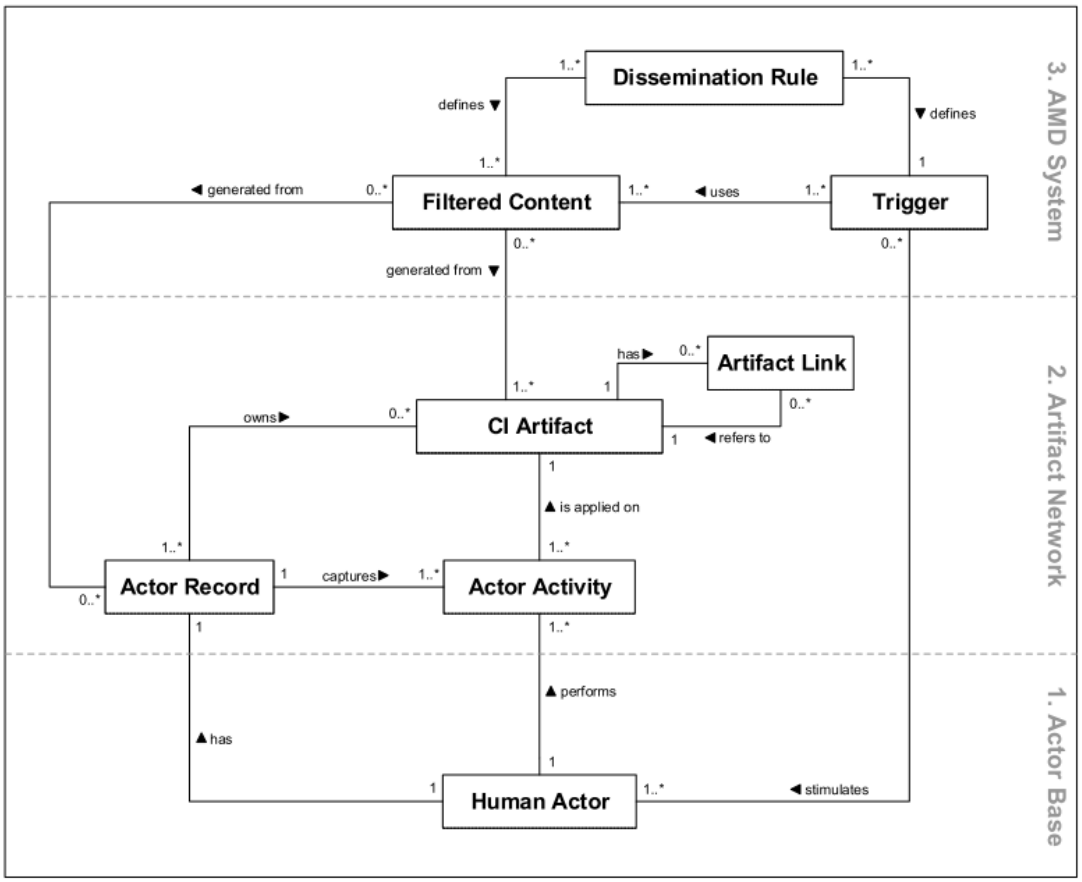

Figure 7.4: Meta-model of the SIS pattern [2] describing a CIS architecture

A main contribution of this thesis is the evolution of a CIS architecture to a CIS farm architecture. Therefore, we extend the meta-model of the SIS pattern, to a *CIS farm meta-model*. In the extended model an additional main component is added, the *Farm Entity Network*, that represents the network of farm entities within a CIS farm. Figure 7.5 shows the extended meta-model, new elements are highlighted in green.

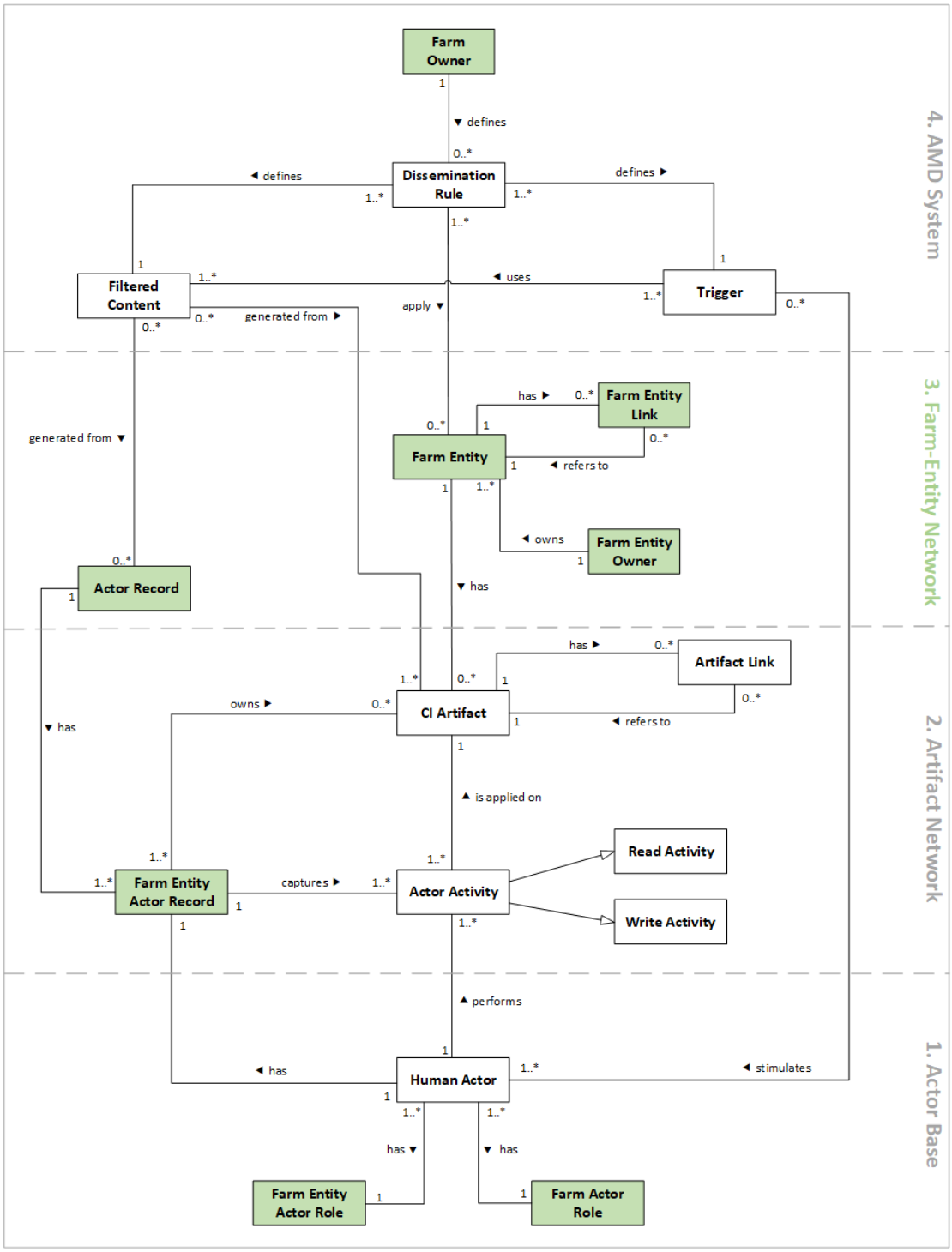

Figure 7.5: Meta-model of a CIS farm architecture

In the following, the new elements and general behavior of the CIS farm architecture are described in more detail.

- A *Human Actor* is a user of the CIS farm.
- A CIS farm consists of several *Farm Entities*, whereby a *Farm Entity* represents a single CIS.
- Every *Farm Entity* has a unique *Farm Entity Owner*, who creates the *Farm Entity* within the CIS farm.
- Each *Human Actor* can be a member of several *Farm Entities*.
- Each *CI Artifact* is explicitly assigned to a *Farm Entity*.
- Only when a *Human Actor* is a member in a *Farm Entity* she can create and edit *CI Artifacts*.
- Within a CIS farm it is possible to link several *Farm Entities* through *Farm Entity Links*.
- *CI Artifacts* can be linked across *Artifact Links* between several *Farm Entities*.
- The *Farm Entity Actor Record* tracks the behavior of each *Human Actor*, e.g., when she edits the content of a *CI Artifact*.
- A *Human Actor* has only a *Farm Entity Actor Record* when she is a member of the corresponding *Farm Entity*.
- Each member of a farm entity has a *Farm Entity Actor Record*, which is connected to the global *Actor Record* of the *Human Actor*.
- The *Farm Owner* defines the dissemination rules for the whole CIS farm, which are executed by the dissemination routine.
- Based on the global *Actor Record* of a *Human Actor* her notifications are generated.
- A *Human Actor* has an individual *Farm Entity Actor Role* in each *Farm Entity*, e.g, she can be an administrator in *Farm Entity A* and a standard user in *Farm Entity B*.
- Every *Human Actor* has a *Farm Actor Role*, which grants him certain permissions at farm level.

7.4 Application of CIS Farm Architecture

Today, for each new CIS with the same structure instance clones are created, where merely a copy of the CIS is filled with new knowledge and evolves more or less independently of the original system. This leads to a variety of limitations, such as high administration effort for the operators and users. Therefore, a CIS farm architecture is useful when similar CIS instances exist with different kinds of knowledge base content, grouped and organized around a specific topic. Operators of a CIS farm can reuse the system capabilities for knowledge with similar structures, e.g., they can effortless create a new CIS in the farm environment. Through a CIS farm architecture several problems of a classical CIS architecture can be solved:

- The administrative effort can be significantly reduced with a CIS farm, since only one system has to be maintained, e.g., updates and bug fixes have to be installed only once.
- A user has only one account for several CIS instances, so called farm entities. After logging in, the user can be identified in all farm entities and can perform various actions.
- Since a user only works with one user account, it is possible to consider several farm entities in the dissemination phase. This may encourage a user to contribute to other related farm entities.
- Since all farm entities are embedded in one environment, it is easy to create a new farm entity. This considerably reduces the administrative burden since not every instance needs to be deployed and maintained individually.
- It is possible to link knowledge across various farm entities. Similar artifacts from different farm entities can be connected within a CIS farm. This makes it possible to build an artifact network across the CIS farm and to find related knowledge faster.
- Integrated data analysis can perform across various farm entities. Therefore, for example, communities can be identified which need additional incentives for further contributions.

In summary, a CIS farm architecture is useful when many instances with a same structure but filled with different knowledge should be created. The designed architecture concept integrates many instances, so-called farm entities, into one environment. For example, Stack Exchange[1] uses the concept of a farm architecture. Stack Exchange is a network of question-and-answer (Q&A) websites, where every single sub-site addresses a certain topic. Within the Q&A network, each user has only one account and knowledge can be linked across different Q&A topic sites.

[1] `https://stackexchange.com`, last visited at 10.02.2019

CHAPTER 8

CIS Farm Evolution Approach

This chapter outlines a process to evolve a CIS platform to a CIS farm. The documentation of all steps of the evolution process is part of the architecture description of a CIS farm. First, the prerequisites are described and afterward the evolution process. Finally, a migration process for the data are described in short form.

8.1 Prerequisites

In chapter 2.5.3 existing software evolution approaches from the literature are described, but during the literature review it turned out that no suitable approach for the evolution of a CIS could be found. Only general statements that facilitate software evolution could be identified. For example, an object-oriented or component-based design improves the reusability [64, 79]. The evolution of a CIS to a CIS farm is basically a discontinuous software evolution, because some essential aspects, such as software architecture and features are changed [68]. A significant change in software architecture is that an enclosing environment is created in which the farm entities are embedded. Furthermore, with a CIS farm architecture, new features are added to the system which brings additional benefits for the stakeholders, such as integrated data analysis across different farm entities, simple creation of new farm entities or notifications for several farm entities.

The CIS-EVO-Farm approach describes the evolution process of a CIS architecture to a CIS farm architecture. This approach can be applied, when a CIS should be evolved into a CIS farm, in which several farm entities are embedded into one environment. To be successful, the following criteria must be fulfilled: (1) access to the CIS code base, (2) access to the CIS data and (3) all farm entities must have the same structure. The CIS-EVO-Farm approach supports software architects to evolve a CIS into a more complex platform, with a system of systems approach.

8.2 Evolution Process

The CIS-EVO-Farm approach describes a light-weight, decision-tree-based process that supports the architectural evolution of a CIS platform into a CIS farm. This approach is designed to assist developers and software architects with a step-by-step guide to evolve a CIS platform into a CIS farm (cf. figure 8.1). The newly designed approach includes the following steps:

1. *Identification*
   The first step is to identify and distinguish the properties of the CIS farm and those of the individual farm entities. In the course of the evolution, components of the enclosing environment (CIS farm) and components of a farm entity are separated. For example, the user management could postpone to farm level, whereas the CI artifact management belongs to a farm entity.

   - *Identify CIS Farm Properties*
     Components that belong to the CIS farm are unique in the entire system and represent the enclosing system (e.g., user-management or administration of the farm entities). The design and functionality of the enclosing system are defined in this step. Later, farm entities are created within the enclosing system.

   - *Identify Farm Entity Properties*
     A farm entity represents an independent CIS in an enclosing environment, in which knowledge is collected. Features that are part of a farm entity are duplicated when a new farm entity is created. Each farm entity should have its own scope in which CI artifacts can be created and edited.

2. *Characterize User Groups*
   During the evolution, the user groups of the CIS farm and the farm entities have to be characterized. An actor has an individual role in the CIS farm and in each farm entity in which she is a member. Furthermore, it has to be determined how a user can become a member of a farm entity (e.g., by invitation).

3. *Determine the Permission System*
   Each defined user group gets certain rights, that allow or forbid to perform specific actions in the system. For example, only an administrator should be able to delete farm entities.

4. *Evolve the System Architecture*
   The architecture is adapted to the new requirements of a CIS farm. Some components of the legacy system have to be moved to farm-level (like user-management), others can remain in the farm entities (e.g., creation of new CI artifacts).

5. *Design Database Schema*
   Once all the important features and components of the CIS farm are identified, the

database schema can be adapted. It must be ensured that a new farm entity can be quickly and easily incorporated and cross-system components are independent of farm entities.

6. *Implementation*
   After the conception phase of the CIS farm is completed, the evolution of the system can start.

   - *CIS Farm*
     First, the fundamental structure of the enclosing system must be created. Above all, the administrative activities of the legacy system are abstracted (like user-management) and the concept to manage the farm entities is implemented. The majority of the dissemination phase is scheduled on farm level, the execution of the rules is done in the farm entities.

   - *Farm Entity*
     In this development phase, most functions of the legacy system are reproduced, and the new cross-functional features of farm entities are implemented. The aggregation of knowledge occurs in the farm entities since a farm entity usually provides an overview of a specific topic.

7. *Verification of the CIS Farm*
   To ensure that the developed CIS farm operates properly and satisfactorily, several tests should be performed. Particular attention should be paid to the interaction of the aggregation and dissemination phases. If, after verification, the result shows that the CIS farm is not operating correctly, a new iteration beginning at the architecture evolution step is necessary. However, if everything works satisfactorily, the last step can be performed.

8. *Migrate Data to the CIS Farm*
   Finally, all data from various CIS instances have to be migrated to the new designed CIS farm. A particular challenge is, that some data has been duplicated in several instances (like user accounts). After migrating the data of an instance, duplicates must be identified and deleted if found. Besides, the IDs of the referenced data must be set correctly.

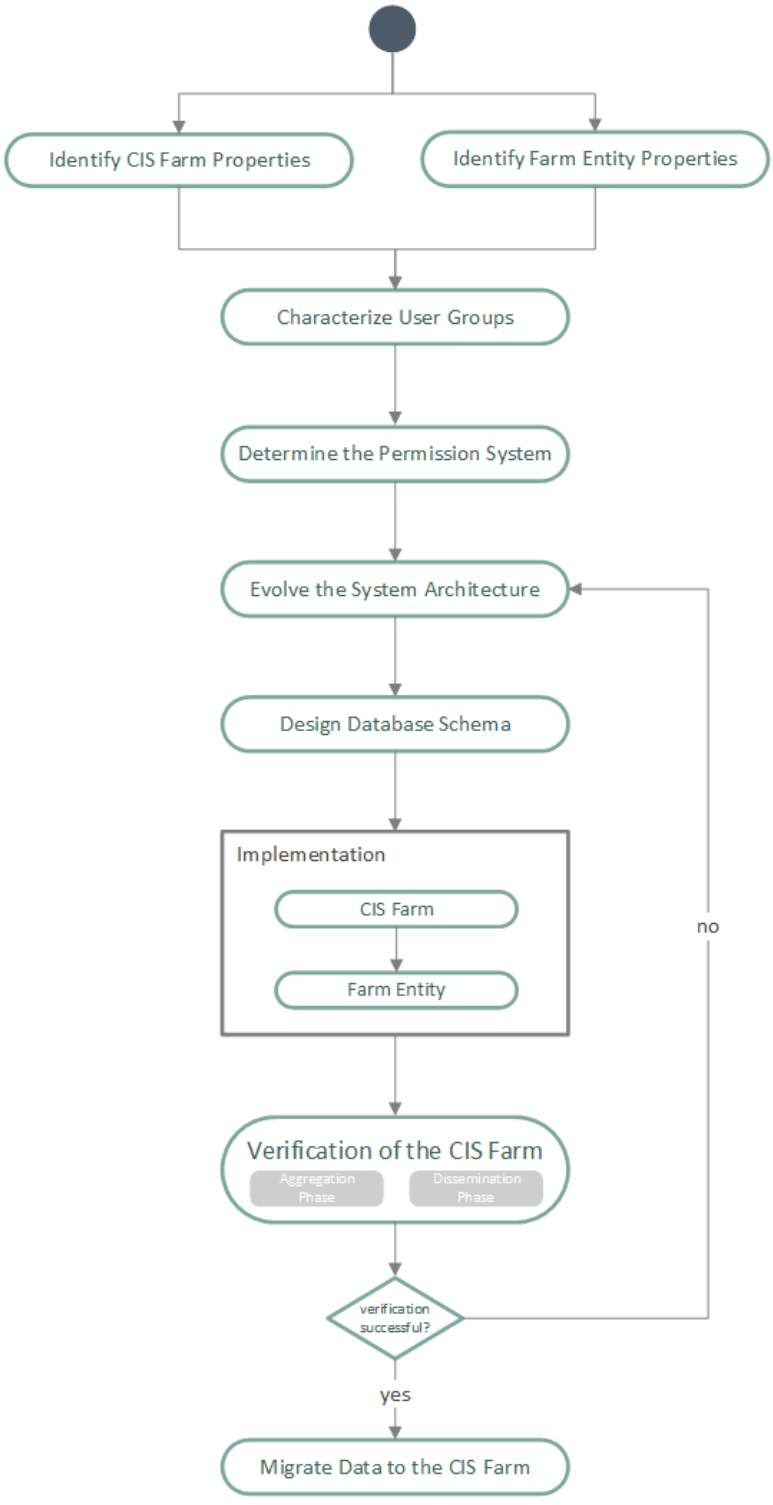

Figure 8.1: Overview of the CIS-EVO-Farm approach

8.3 Data Migration Process

Based on our assumption, that several CIS platforms existed in parallel with the same code base, but filled with different data and knowledge, the following data migration process can be executed to migrate all data into one environment. The data migration process contains 6 steps, where step 3 to 6 must be repeated as many times as farm entities should be created:

1. *Identify data to migrate*
   All data to be transferred must be identified before the actual migration can be initiated. It is possible, that some data will not be transferred, because they are not required or incompatible to the new implementation.

2. *Merge farm-wide values*
   Some data exist several times in the CIS instances, but these need to be unified in the CIS farm, such as user accounts.

3. *Create new farm entity*
   A new farm entity needs to be created in which, among other things, the content of the CI artifacts can be integrated.

4. *Add specific values to the farm entity*
   Several specific farm entity values need to be set, such as the administrator or the visibility settings of the farm entity.

5. *Assign users to the farm entity*
   All transferred or merged users have to be added to the farm entity as members, so they can continue their work in the new environment.

6. *Integrate data to the farm entity*
   Concluding, all remaining values can be transferred to the farm entity, such as the content of the CI artifacts.

After the evolution and migration process, the CIS farm and data can immediately be accessed by the users without a new user account or manual transfer of the data. The community can directly take full advantage of the CIS farm, such as link knowledge across various farm entities or create effortless a new farm entity. Besides, the individual CIS instances can be archived as they are no longer needed.

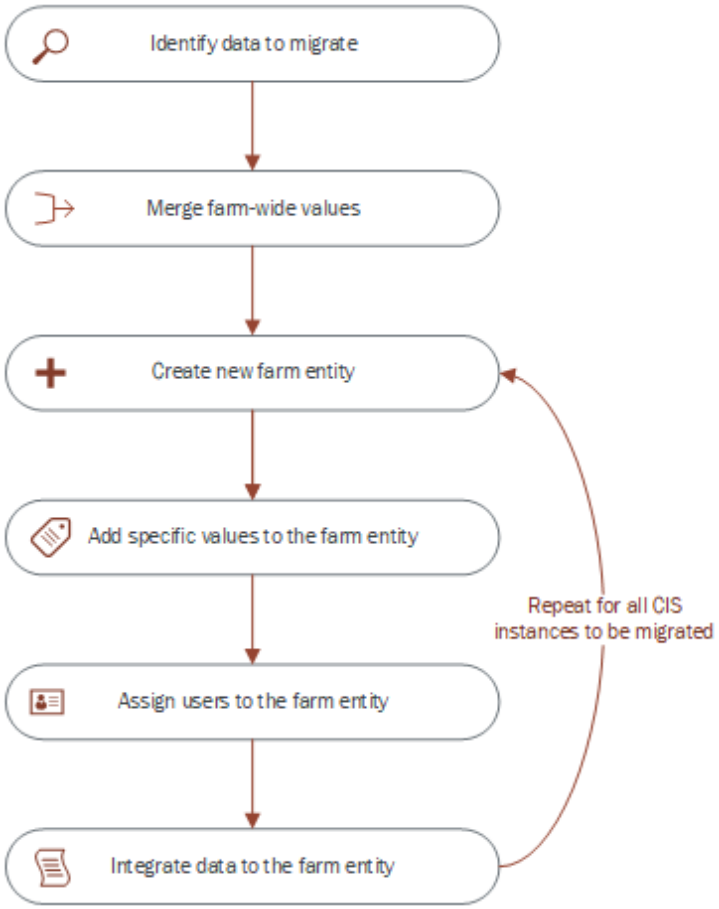

Figure 8.2: Migration process

CHAPTER 9

Application of CIS Farm Architecture Design Approach

This chapter describes the application of the proposed CIS farm architecture design approach by implementing a prototypical CIS farm system. First, the initial state is briefly discussed, then stakeholders are described. Afterwards, typical use cases of the Glossary Farm are presented, before the implementation of the system is explained. Finally, some challenges during the development are discussed.

9.1 Initial System

As described in section 5.4.4, there exist several challenges and limitations when operating standalone Glossary Platforms. A new system instance has to be created for every glossary project, which results in a significant effort for the system management and configuration, e.g., updates and bug fixes have to be executed separately for each system. Furthermore, the creation of a new instance is expensive and time-consuming, because a new instance has to be deployed by an administrator. In addition, a user who is involved in more than one project has to create and manage several accounts.

Figure 9.1 illustrates the current state of the architecture design of the Glossary Platform. This approach has various limitations, such as many standalone instances, several accounts for one researcher and no cross-instance linking of knowledge. To address the mentioned limitations of the Glossary Platform the CIS farm architecture approach is an applicable solution because all farm entities are integrated into one system environment.

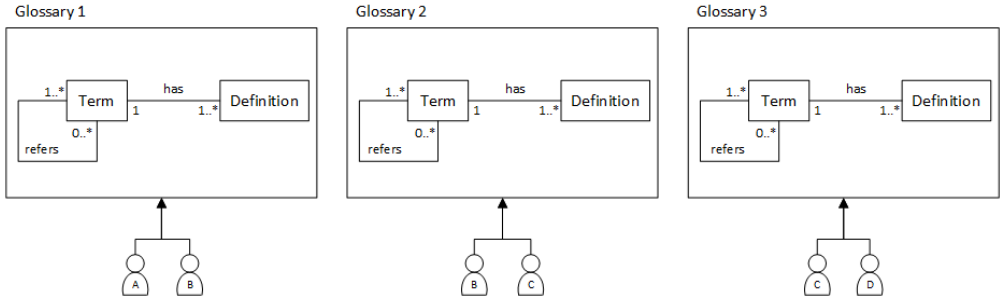

Figure 9.1: Architecture design of the Glossary Platform

9.2 Evolution to CIS Farm Architecture

The purpose of the Glossary Platform evolution is to verify the designed CIS farm architecture and the evolution approach in practice. Besides, the users of the Glossary Platform will profit from the new features of the Glossary Farm as they can now collaborate across various farm entities in one platform.

9.2.1 Target System

The final state after the evolution of the Glossary Platform into the Glossary Farm is shown in figure 9.2. New components are painted green in the illustration. Within the Glossary Farm, several glossaries (farm entities) are embedded into a common environment. Through the enclosing system, users get an overview of all existing glossaries and can switch between them easily. In addition, terms can now be linked across instances. One of the greatest advantages of the Glossary Farm is the minimized administrative effort. Firstly, creating new instances is time-saving and uncomplicated. Secondly, users only possess one account.

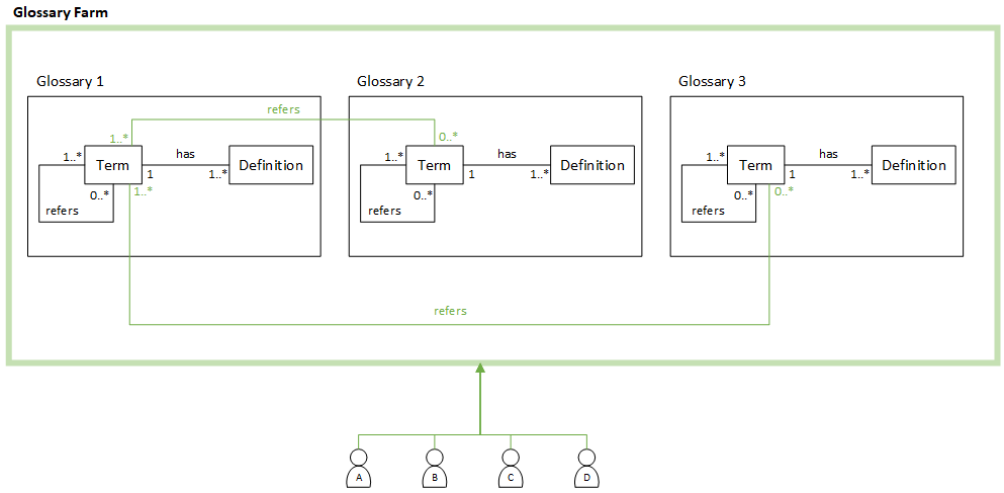

Figure 9.2: Architecture design of the Glossary Farm

9.2.2 Stakeholders

The Glossary Farm has a wide variety of stakeholders. It is possible that some stakeholders are embodied by the same person, for example, a *Farm User* is a member of various glossaries and becomes in these a *Glossary User*. The key stakeholders of the Glossary Farm are:

- *Platform Operator(s)*, who define the purpose of the Glossary Farm and make the service available to users.
- *Farm Admin(s)*, who take over the administration of the Glossary Farm, such as invite new users and create, share or deactivate glossaries.
- *Farm User(s),* who have access to the Glossary Farm. Each farm user can explore all public glossaries and has an individual list of glossaries where she holds a membership.
- *Glossary Admin(s)*, who manage a particular glossary. For example, she can add new members to the glossary or remove existing ones.
- *Glossary Moderator(s)*, who have access to a particular glossary and perform lightweight administrative activities, such as delete terms.
- *Glossary User(s)*, who have access to a particular glossary and contribute to that.
- *Unregistered User(s)*, who can view all public glossaries, but cannot contribute to any glossary.

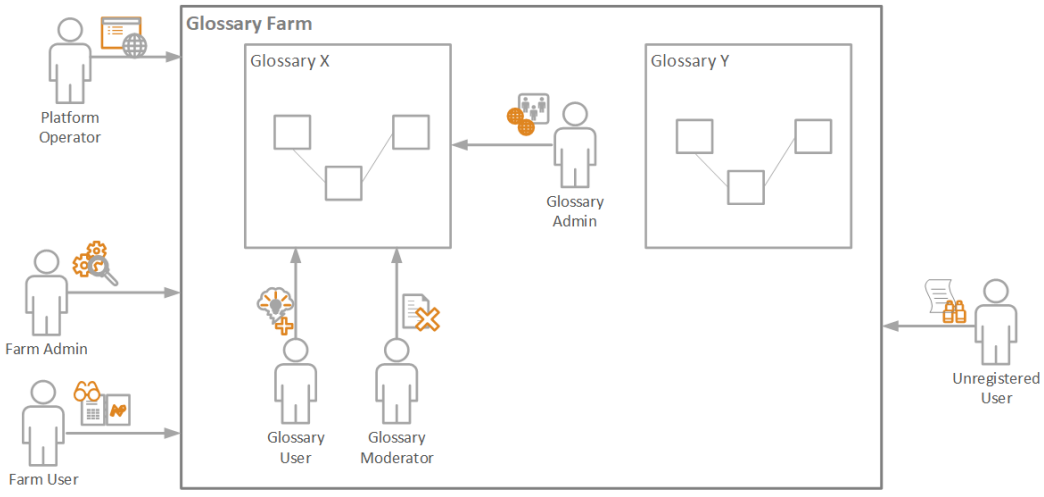

Figure 9.3: Glossary Farm stakeholders

9.2.3 Main Use Cases

In the following, the main use cases of the Glossary Farm are described. The use cases 1-12 and 32-34 are based on farm level, the use cases 13-31 are always executed within a glossary. Artifacts can only be created and edited by members of a glossary. Unregistered users can view artifacts from public glossaries, artifacts of private glossaries can only be considered of members. After listing the use cases, it will be illustrated in more detail which stakeholders are allowed to do what.

1. **Create Glossary**. Create a new glossary in the farm environment. Depending on the farm actor role, either the glossary is immediately created or a creation process is initiated. *Farm Admins* can create and activate glossaries immediately. *Farm Users* can only submit requests for a new glossary, which can then either be accepted or rejected by a *Farm Admin*.

2. **Change Glossary Status**. A glossary has a unique status, depending on the status users can view the glossary or not (activated, deactivated). The status of glossaries can only be changed by *Farm Admins*.

3. **Edit Glossary**. The properties of glossaries can be edit, such as visibility, description, identifier, assigned categories or administrator of the glossary. *Farm Admins* can edit all glossaries, *Glossary Admins* only assigned glossaries.

4. **Create User Account**. An *Unregistered User* can create a Glossary Farm account. Only with an account, a researcher can become a member of a glossary and create contributions.

5. **Invite a new User**. A *Farm Admin* can invite new users to the Glossary Farm. Basically, there is no difference to the use case *Create User Account*, except that these users follow an invitation.

6. **Add User to Glossary**. A *Farm Admin* or *Glossary Admin* can add a registered user to a glossary. Only when a user is a member of a glossary, she can create and edit contributions.

7. **Remove User from Glossary**. A *Farm Admin* or *Glossary Admin* can revoke a user's membership in a glossary, afterward the user can no longer collaborate to the glossary.

8. **Create Category**. *Farm Admins* can create new categories, which can then be assigned to glossaries.

9. **Delete Category**. A *Farm Admin* can delete an unused category.

10. **Show Categories**. Registered and unregistered users can view all categories with

the assigned public glossaries, registered users also see private glossaries in which they are members. Thereby relevant glossaries can be found faster by a user because they are grouped.

11. **View Statistics**. Various statistics can be generated from the activities of users in the Glossary Farm, such as total term count, contributions in the last week and month and contributions per user. The statistics may display different values for users since activities of private glossaries are only included in the statistics for *Farm Admins* and members of these glossaries.

12. **View Glossary Statistics**. The statistics can also be displayed only for individual glossary entities.

13. **Create Term**. Members of a glossary can create new terms, which are the core elements of a glossary.

14. **View Term**. Everyone can display the details of a term. Terms of private glossaries can only be displayed by members.

15. **View Term Overview**. Everyone can view a summary of all the terms of a glossary.

16. **Edit Term**. Terms can be edited by members of a glossary.

17. **Add Synonym to a Term**. If a term has an interchangeable term in the Glossary Farm, it is possible to hyperlink these.

18. **Add Related Term to Term**. If a term has a related, but not identical term in the Glossary Farm, it is possible to connect these.

19. **Delete Term**. *Farm Admins*, *Glossary Admins* and *Glossary Moderators* can delete terms of a glossary. *Glossary Users* can only delete their own created terms.

20. **Term Search**. Registered users can search for a term in the Glossary Farm or an individual glossary entity.

21. **Add Definition to a Term**. Members of a glossary can add a definition to a term. Each term have at least one definition, that describes the term more precisely and specify its clear meaning in a defined context.

22. **Show Definitions of a Term**. A term is displayed with all associated definitions.

23. **Edit Definition of a Term**. A definition of a term can be edited by members of the glossary.

24. **Remove Definition of a Term**. *Farm Admins*, *Glossary Admins* and *Glossary Moderators* can delete definitions of a term. *Glossary Users* can only delete their own created definitions.

25. **Create Term Comment**. Members of a glossary can create comments to discuss information about a term.

26. **Show Term Comments**. Registered users can view all comments about a term.

27. **Change Term Quality Level**. Members of a glossary can change the quality level of a term. Validated terms contain only reviewed information.

28. **Change Definition Quality Level**. Members of a glossary can change the quality level of a definition. Validated definitions contain only reviewed information.

29. **Show Term History**. Changes on a term and its associated definitions are logged, such as editor or creation time of a new definition. The history of a term can be displayed in addition to the stored knowledge of a term.

30. **Upvote Definition**. Members of a glossary can upvote a definition of a term, to show that the definition is meaningful and understandable.

31. **Downvote Definition**. Members of a glossary can downvote a definition of a term, to show, e.g., that the definition is not meaningful or confusing.

32. **Send Global Digest**. The global digest informs users once a week about recent changes in the glossary and shows interesting stuff, such as trending terms and definitions (detected by the number of views), or artifacts, which needs validation.

33. **Send Personal Digest**. The personal digest shows once a week recent changes to own contributions and suggests similar terms and definitions for further inspection (based on views, tags, synonyms and related terms)

34. **Send Ranking Summary Digest**. The ranking summary is sent monthly and provides information about the number of changes per user, the one with the most contributions gets the first place. The ranking summary should encourage the users for further contributions.

Main Use Cases of the Farm Admin

The main use cases of the stakeholder *Farm Admin* are illustrated in figure 9.4. This stakeholder mainly takes care of administrative activities within the Glossary Farm. On the one hand, the *Farm Admin* manages the users, invites new ones or deletes inactive users from the system. On the other hand, she creates new glossaries, change the glossary status and edit glossary specific settings.

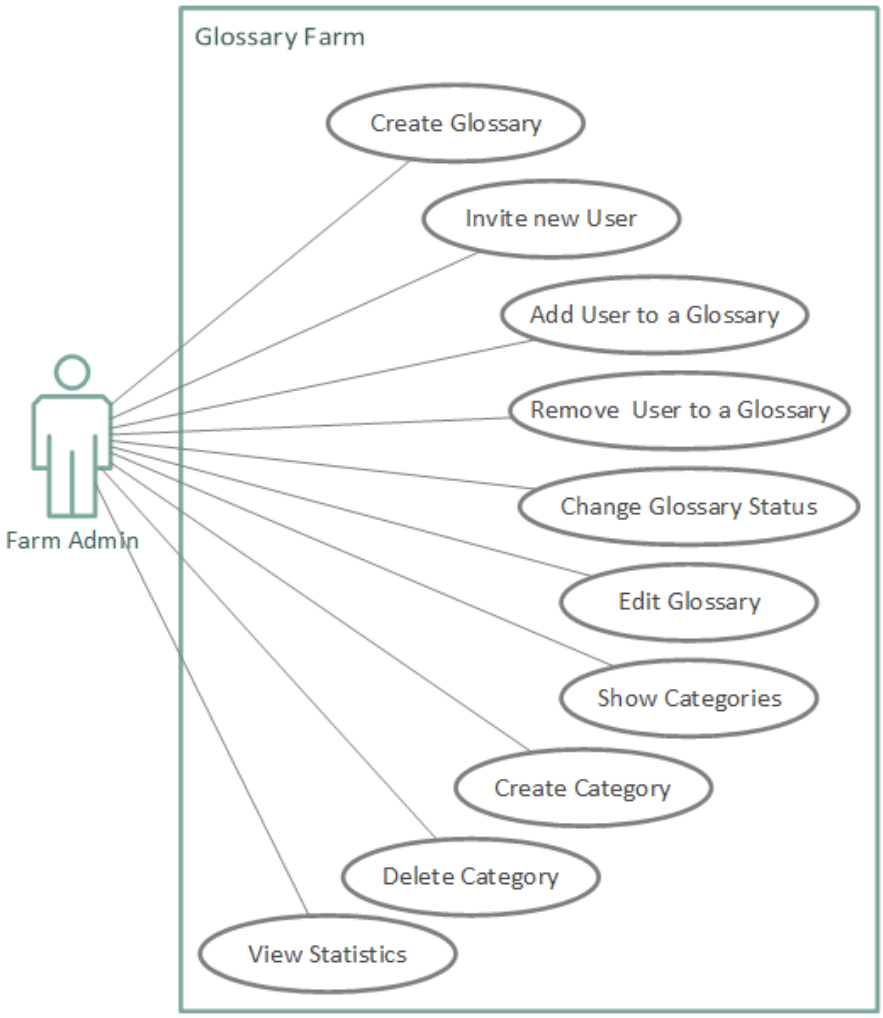

Figure 9.4: Main use cases of the Farm Admin

Main Use Cases of the Glossary Admin, Moderator & User

In the following figure 9.5 the use cases of the stakeholders *Glossary Admin*, *Glossary Moderator* and *Glossary Admin* are shown. All use cases of the *Glossary User* concern the collection, verification and viewing of knowledge. For example, create a term, update a definition, validate a term or view a term and its definitions. In contrast to the *Glossary User*, the *Glossary Moderator* can also delete contributions from others. She monitors all changes in terms and definitions and deletes incorrect or inappropriate contributions. The *Glossary Admin* can additionally perform administrative activities in her glossary, such as add a user to the glossary or update the glossary description.

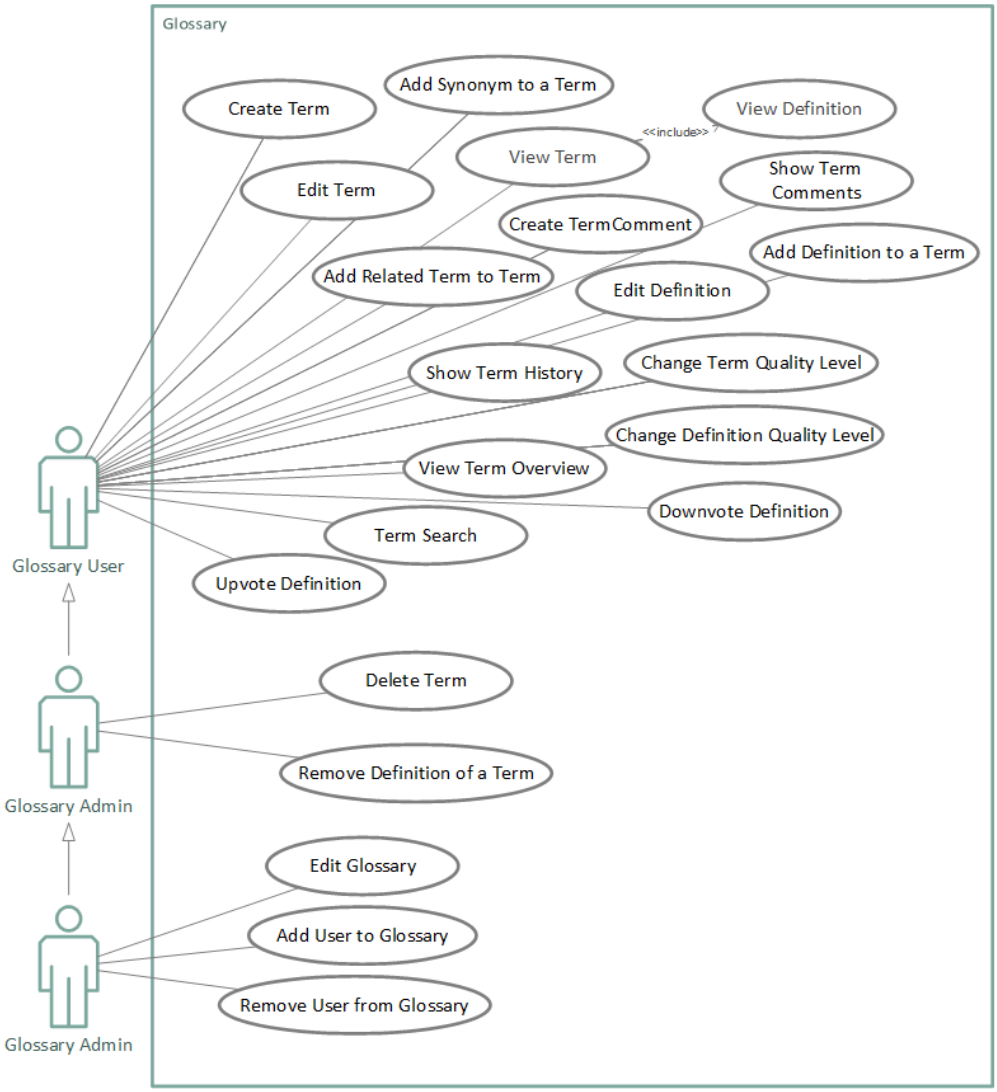

Figure 9.5: Main use cases of the Glossary Admin, Moderator and User

9.2.4 Permission System

To clarify which stakeholders are allowed to do what, the permission system of the Glossary Farm is described in more detail. The permission system of a glossary farm is role-based, the authorization within a glossary depends on the global and local role of a user. *Farm Admins* are allowed to perform all actions, regardless of whether they are members of a glossary or not. *Glossary Admins* can perform all activities inside their glossaries, while *Glossary Users* cannot delete artifacts of others. The role system is described in more detail in figure 9.6.

User	Glossary-Farm Role
User 1	Farm Admin
User 2	Farm User
User 3	Farm User
User 4	Farm User

User	Glossary A Role	Glossary B Role
User 1	-	-
User 2	Glossary Admin	Glossary Admin
User 3	Glossary User	Glossary Moderator
User 4	-	-

Glossary A - public, Glossary B - private

✓ allowed
✗ denied
(x) only allowed if creator
(-) not all information visible

Action	User 1		User 2		User 3		User 4		unregisterd User	
	Glossary A	Glossary B	Glossary A	Glossary B	Glossary A	Glossary B	Glossary A	Glossary B	Glossary A	Glossary B
View Glossary	✓	✓	✓	✓	✓	✓	✓	✗	✓	✗
View Term	✓	✓	✓	✓	✓	✓	✓	✗	✓	✗
Search Term	✓	✓	✓	✓	✓	✓	✓	✗	x	✗
Create Term	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗
Edit Term	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗
Delete Term	✓	✓	✓	✓	(x)	✓	✗	✗	✗	✗
Validate Term	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗
View Tag	✓	✓	✓	✓	✓	✓	✓	✗	✓	✗
View Statisitc	✓	✓	✓	✓	✓	✓	✓	✗	✓	✗
within a term										
Add Definition	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗
Edit Definition	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗
Delete Definition	✓	✓	✓	✓	(x)	✓	✗	✗	✗	✗
Validate Definition	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗
View Comments	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗
Add Comment	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗
View History	✓	✓	✓	✓	(-)	(-)	(-)	✗	(-)	✗
Upvote Definition	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗
Downvote Definition	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗
Mark-as-best Definition	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗

Figure 9.6: Permission system of the Glossary Farm

9.2.5 Evolution Process

Figure 9.7 illustrates the Glossary Platform evolution to a Glossary Farm. The application was evolved on the basis of the CIS-EVO approach. The general description of the CIS-EVO approach can be found in section 8.2, each step of the CIS-EVO approach is described in a separate section. First, the requirements for the Glossary Farm were derived before the implementation was started. Afterward, the system was evaluated to check if the requirements were implemented correctly.

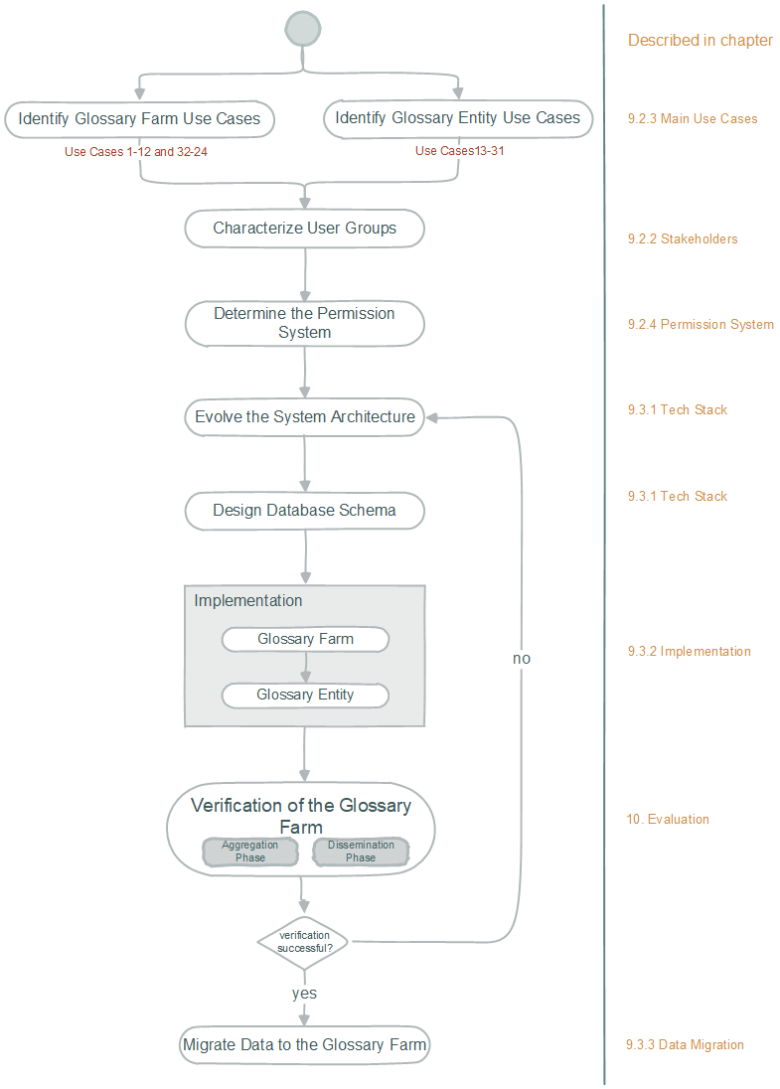

Figure 9.7: Evolution process to the Glossary Farm

9.3 Development of Prototype

This section describes the tech stack and the data model of the Glossary Farm. In addition, the implementation of CIS farm characteristics is presented and how we migrated the data from the Glossary Platform instances to the Glossary Farm.

9.3.1 Tech Stack

The Glossary Farm is a web application that is accessible via a web browser, either on a PC or mobile device. The platform is implemented in Ruby on Rails with a PostgreSQL database. We used RubyGems[1]as package manager to manage and install the needed libraries (called *gems* in the ruby environment). Some well-known ruby libraries that we use are the gem *devise* for the user management, the gem *pundit* for the user authorization and the gem *audited* for versioning. For designing the user interface we used Bootstrap[2], a front-end responsive framework, in combination with Font Awesome[3] for the icons. For the source code management we use a git repository on GitHub[4] and a scrum development process with two weeks sprints. The application is deployed on Heroku[5], where researchers can access the Glossary Farm.

Figure 9.8 illustrates the data model of the Glossary Farm, new tables and attributes are highlighted in green. The data model is an evolution of the Glossary Platform data model, which is presented in 5.4.3. The presented data model contains not all tables and attributes, it only illustrates a selection of the most essential components. Following, a brief description of the new tables and their relationships:

- The table *glossaries* contains the farm entities, which were prior standalone CIS instances. Each entry in the table has a unique identifier, to select the glossary via the URL in the web browser.
- The table *glossary_users* contains the assignment of the members to a glossary.
- The table *categories* stores all the categories of the Glossary Farm.
- The table *glossary_categories* contains the assignment of the categories to a glossary.
- To the tables *activities*, *terms* and *taggings* a new attribute was added. Through the attribute *glossary_id* the corresponding of a glossary data sets can found.

[1] `https://rubygems.org`, last visited at 19.02.2019
[2] `https://getbootstrap.com`, last visited at 19.02.2019
[3] `https://fontawesome.com`, last visited at 19.02.2019
[4] `https://github.com/amusil/glossary-farm`, last visited at 19.02.2019
[5] `http://glossary-farm.herokuapp.com`, last visted on 1.12.2018

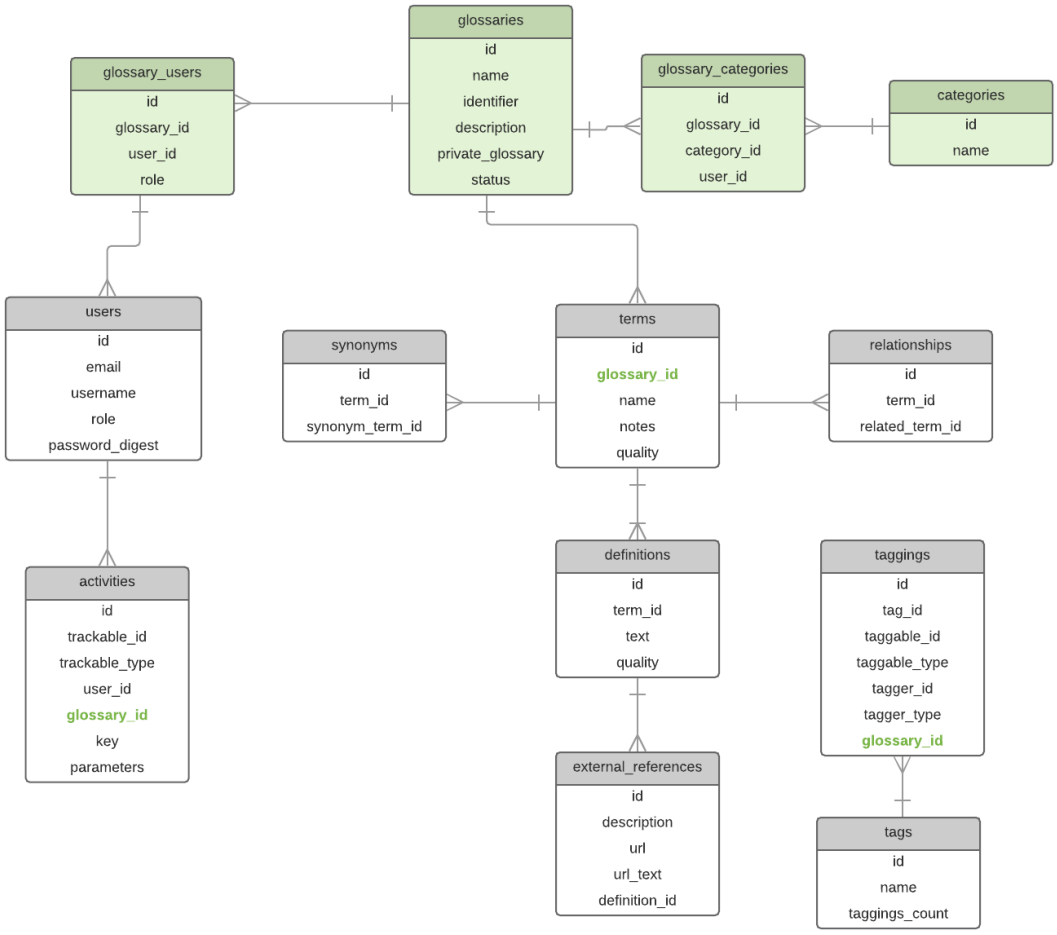

Figure 9.8: Data model of the Glossary Farm

9.3.2 Implementation

In the following, we describe how we implemented the system characteristics of a CIS farm (compare section 6.1) in the Glossary Farm. All fourteen properties of a CIS farm have been successfully implemented.

1. *CIS farm administrator*
   The Glossary Farm has a user role *administrator*. Each user with this role can administrate the farm, such as create or activate glossaries. Figure 9.9 illustrates the glossaries management overview, to which only users with the role *administrator* have access.

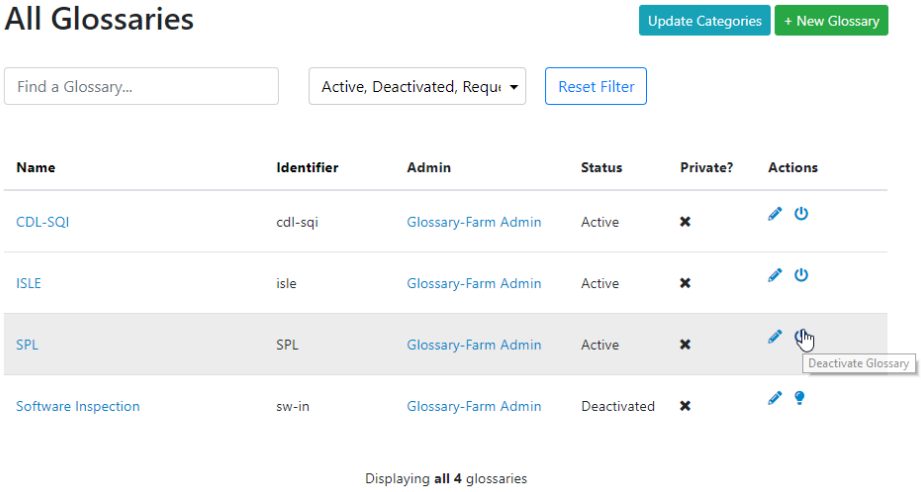

Figure 9.9: Screenshot: Glossaries management overview

2. *Global unique user account*
   Each user has a unique user account in the Glossary Farm. Figure 9.10 shows the user management list at farm level. To this view only users with the role *administrator* have access.

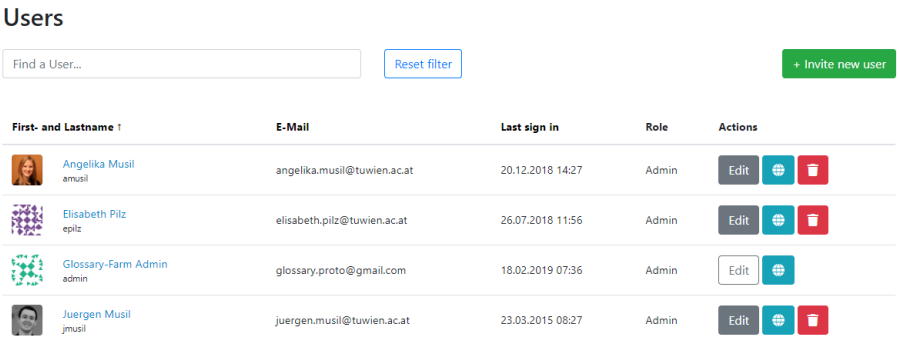

Figure 9.10: Screenshot: Users management overview

3. *Global individual user profile*
   Each user has a global individual user profile, as shown in figure 9.11.

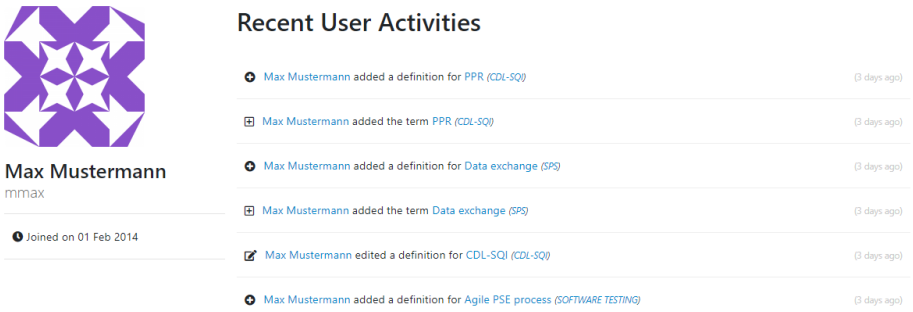

Figure 9.11: Screenshot: User profile view

4. *System-wide permissions*
   Users of the Glossary Farm have individual rights on the global level, the user roles *Farm Admin* and *Farm User* are implemented. Users with a role admin posses more rights, e.g., they are allowed to delete other user accounts.

5. *Process to create new farm entity*
   Farm Admins can create a new farm entity immediately. Requests of *Farm Users* for a new glossary entity must first be approved by an *Farm Admin*. Figure 9.12 illustrates the request of a *Farm User* to crate a new glossary and figure 9.13 shows the view of a *Farm Admin* to crate a new glossary. Only after a *Farm Admin* has accepted the request for a new glossary other users can access the glossary.

Figure 9.12: Screenshot: Request new glossary by a Farm User

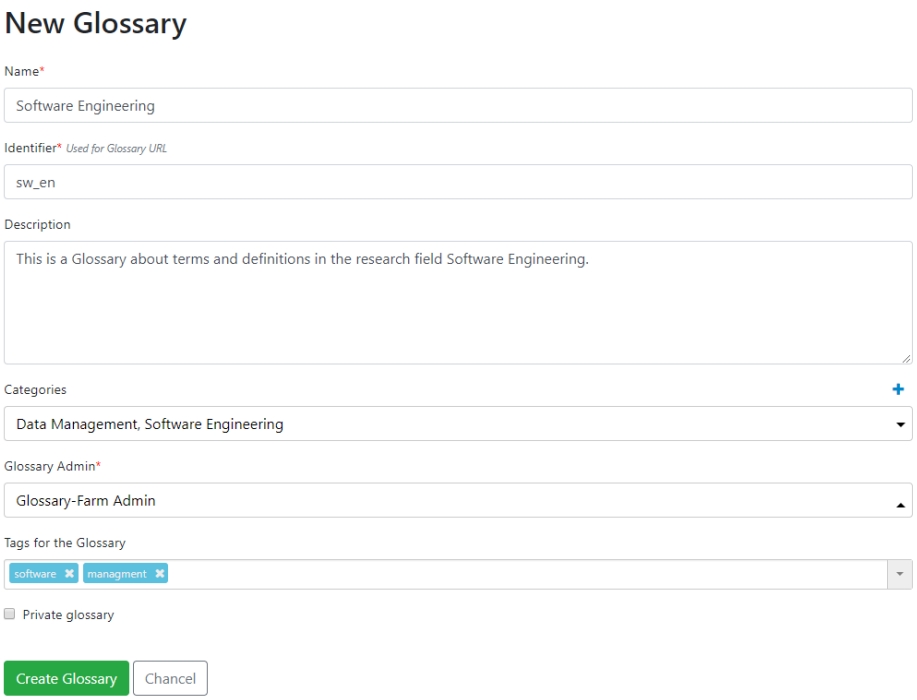

Figure 9.13: Screenshot: Create new glossary by a Farm Admin

6. *Classification of farm entities in categories*
   Glossary entities can be subdivided into categories, to identify faster their area of application. Figure 9.14 shows all categories of the Glossary Farm and figure 9.15 presents all glossaries which are associated with the category *Software Engineering*.

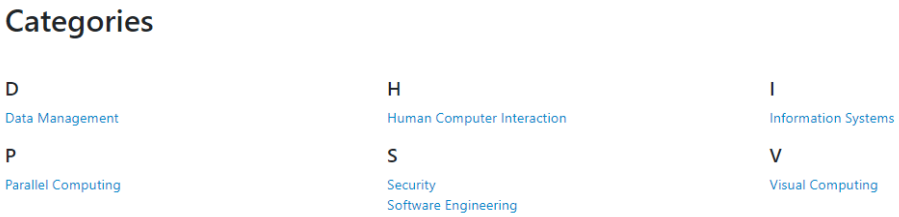

Figure 9.14: Screenshot: Categories view

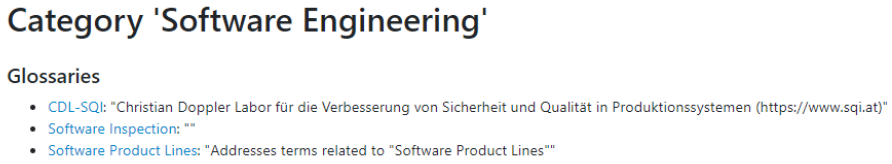

Figure 9.15: Screenshot: Detail view of the category *Software Engineering*

7. *Visibility of the farm entities*
   Glossary entities have either public or private visibility. Everyone, included unregistered user, has access to public glossaries and can view the artifacts. Private glossaries can only be viewed by members of this glossary. Figure 9.9 shows also the visibility of the glossary entities, see penultimate column.

8. *User assignment to farm entities*
   Only members of a glossary entity get write permissions, others have just read rights. The write permission allows users to create new terms or definitions, or to link a term with another. Figure 9.16 shows the view of a *Glossary Admin* about members. In this view she can either add or remove users.

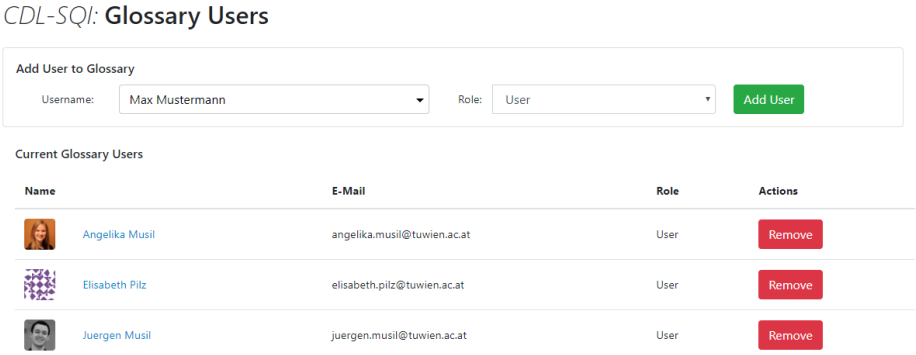

Figure 9.16: Screenshot: User management of a farm entity

9. *Individual permission for every farm entity*
   Users have different rights in each glossary entity, where they are members. In one glossary they can be the administrator, in others they can act as a moderator or as a standard user.

10. *Artifacts belong to a farm entity*
    Each term belongs precisely to one glossary. Figure 9.17 shows the term overview of a glossary entity.

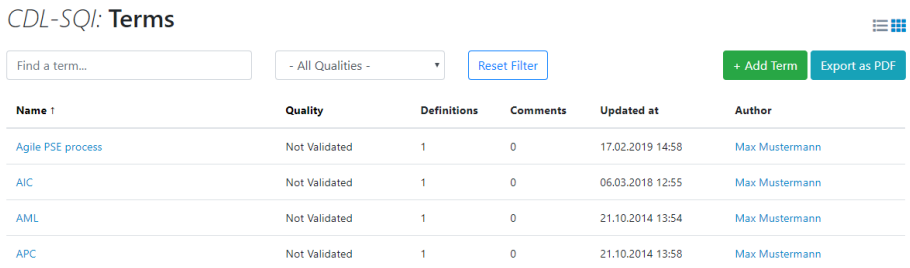

Figure 9.17: Screenshot: Term overview of a glossary

11. *Create links between different farm entities*
    Similar terms from different glossaries can be connected within the Glossary Farm. Figure 9.18 illustrates links between different terms in various glossaries, the glossary is in parenthesis.

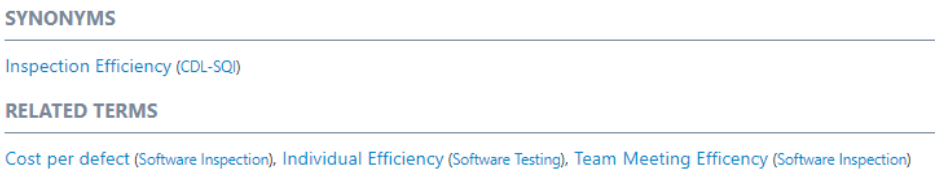

Figure 9.18: Screenshot: Synonyms and related terms

12. *Artifact Search*
    Terms are searchable across all glossary entities. Figure 9.19 shows the search result for a term.

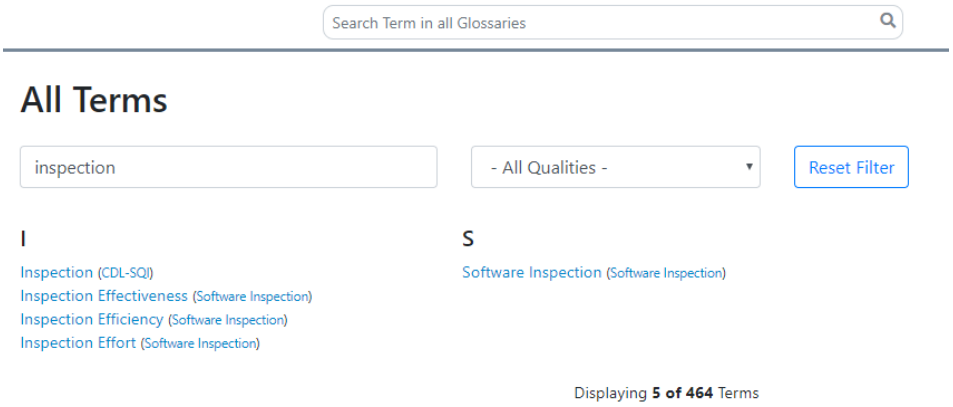

Figure 9.19: Screenshot: Search result for a term

13. *Notification system*
    The Glossary Farm has a global notification system to encourage users to make further contributions. The figure 9.20 illustrates a global digest notification. The global digest informs users once a week about recent changes in the Glossary Farm and shows interesting stuff. The mail provide a separate section for each glossary, where the user is a member.

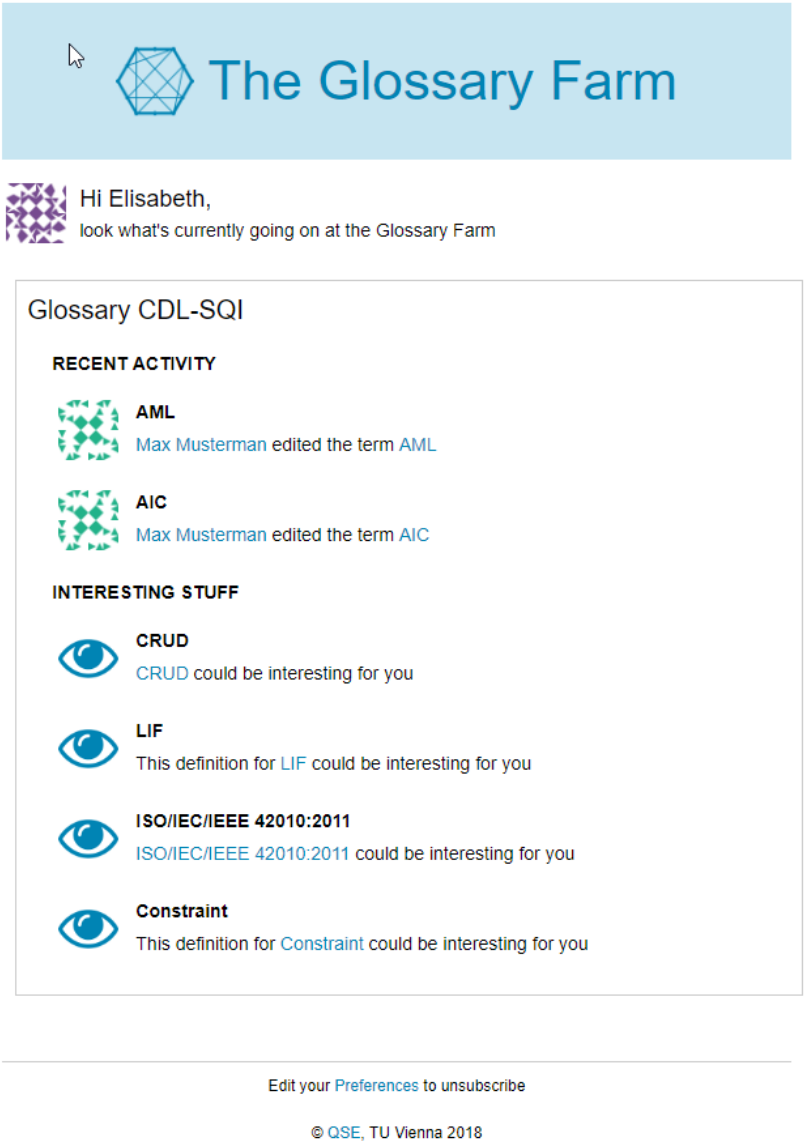

Figure 9.20: Screenshot: Weekly Global Digest

14. *Individual notification per farm entity*
Notifications mails are structured in sections. For each glossary, where the user is a member, a separate section is created. Furthermore, the users get a notification if a term or definition of her gets edited.

15. *Notification setting for each farm entity*
Notifications are customizable for each glossary entity. This makes it possible for users to receive all, only certain or no messages for a glossary entity. Figure 9.21 and 9.22 illustrates the global and activity notification settings, for each glossary,

in which the user is a member, a row is shown.

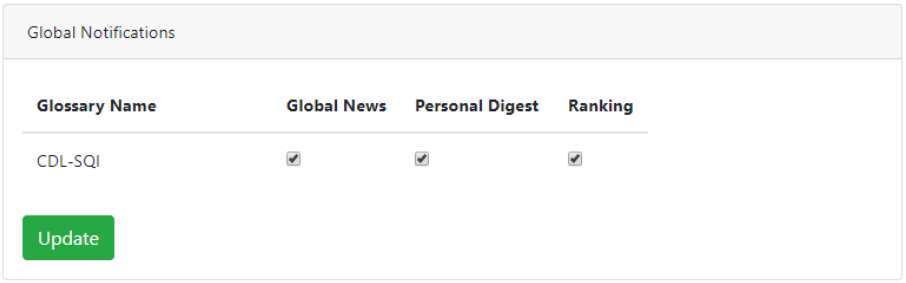

Figure 9.21: Screenshot: Global notification settings

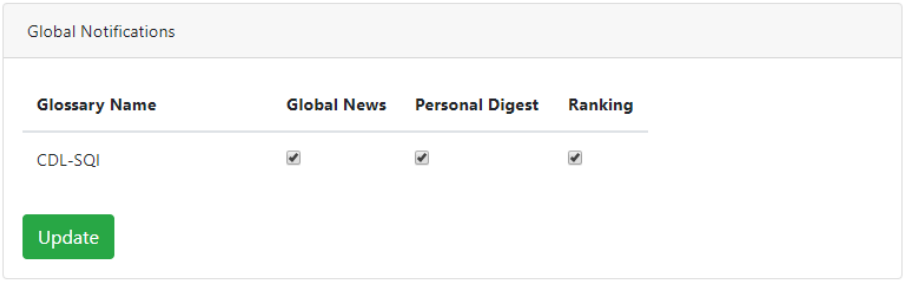

Figure 9.22: Screenshot: Activity notification settings

9.3.3 Data Migration

We used Pentaho Data Integration[6] to migrate the data to the new Glossary Farm. Pentaho Data Integration consists of a core data integration engine and GUI applications that allow users to define data integration jobs and transformations. For each table, we created a new transformation in Pentaho and the necessary steps to migrate the data to the corresponding Glossary Farm table. All these transformations were executed within a Pentaho job (compare figure 9.24).

Figure 9.23 shows the transformation steps for the table *terms*. It should be noted that the *id* for the creator, editor and validator had to be overridden in the Glossary Farm table. For each email address from a Glossary Platform, the *id* of the corresponding email address was searched in the Glossary Farm. This is necessary because the users have only one account in the Glossary Farm and their data has to be mapped to the corresponding account.

[6] `https://community.hitachivantara.com/docs/DOC-1009855-data-integration-kettle`, last visited at 19.02.2019

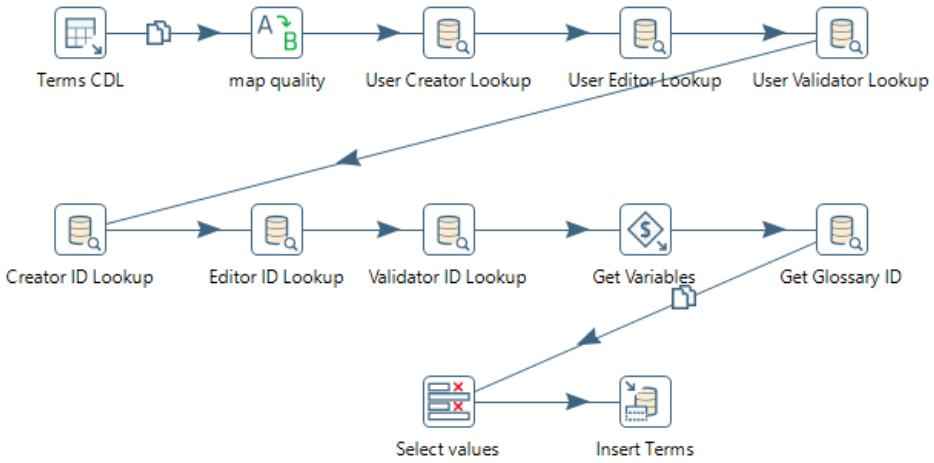

Figure 9.23: Transformation in Pentaho for the table *terms*

The process of migrating all necessary data from a Glossary Platform instance to the Glossary Farm is shown in figure 9.24. This job includes all transformation steps to transfer the data into the Glossary Farm with correct relationships. This migration job was repeated for each glossary to be migrated, the glossary specific data were set in the step *Set Variables* - for example, the *name* and *description* of the glossary or the *id* of the administrator.

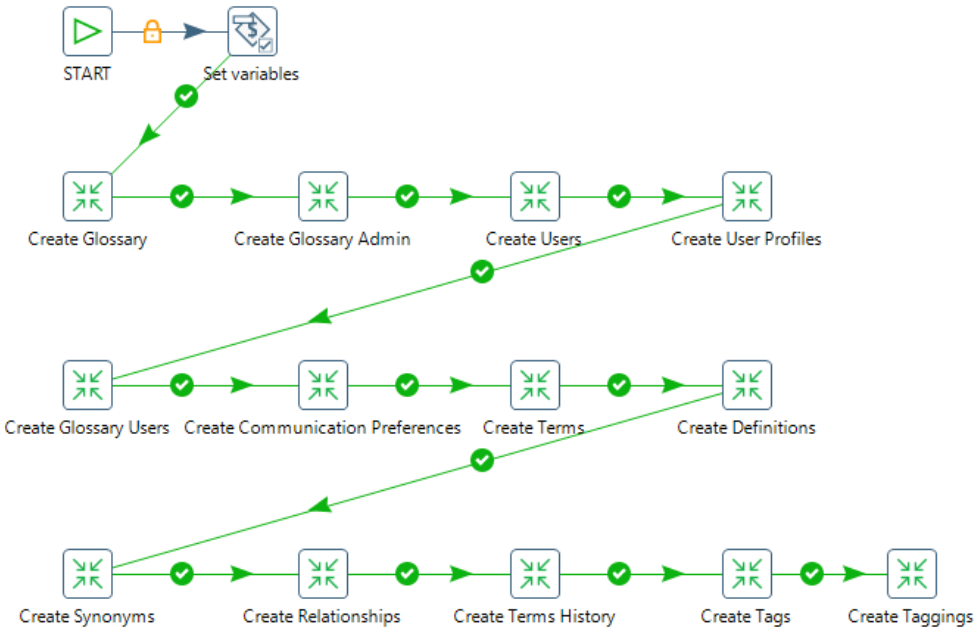

Figure 9.24: Migration job in Pentaho

9.4 Lessons Learned

The CIS farm architecture description was quite helpful for implementing the Glossary Farm. A CIS farm architecture solves several problems of a classical CIS architecture. Several glossaries can be embedded into one enclosing environment, which offers a variety of advantages. For example, (1) effortless creation of a new glossary, (2) only one account for researches and (3) link knowledge across various glossaries. In addition, the CIS-EVO approach provides useful guidance for the evolution process. We followed the CIS-EVO approach during the evolution and had no significant problems to implement the Glossary Farm. Nevertheless some problems and challenges have arisen during the implementation, which are briefly explained in the following:

- **Obsolete Libraries**
  The initial system was not updated to the latest version of the libraries (gems) for several years, which caused some performance and security issues. It takes a large amount of time to update them to the latest versions and to fix the problems caused by the upgrade process. Sometimes the code was adapted or rewritten to use the libraries optimally or to use new features of these.

- **Access Authorization**
  Major efforts were made to design a good access authorization system. Depending on the visibility of a glossary entity and the role of a user in these glossary entity, different things are displayed or hidden. For example, unregistered users see only activity logs of public glossaries on the dashboard. If a logged-in user is a member of a private glossary, the dashboard is extended with activity logs of this glossary. The implementation was carried out using a library (gem), which checks if a user has sufficient permissions to view or manipulate certain elements.

- **UI Design**
  Since a Glossary Farm represents a system-of-systems, it was essential to create a UI design that allows a clear distinction between farm and glossary entity components. In advance, some fundamental considerations were made, how a clear separation of these components are possible. For this purpose, some mock-ups were designed and afterward compared. Finally, that design was selected where farm components are in the right sidebar and entity specific components in the header.

- **Adapt Dissemination Phase**
  Another big challenge was to adapt the existing dissemination phase (via e-mail) in such a way that only relevant entities for a user are included. A user should only receive e-mail summaries and suggestion for glossaries in which she is a member. To solve this issue, an additional check has been added during the e-mail generation, to determine if a user is a member of a glossary. If so, then the glossary is included in the summary for the user. Therefore, each user will always receive a tailored email report on her glossary memberships.

- **Migration**
  One requirement was that the existing data from the individual glossary instances are migrated into the new farm platform. For each existing glossary instance, a correlating farm entity was created. Since various database tables have changed, no one-to-one transmission to the new system was possible. In particular, it was necessary to ensure that user accounts are only created once in the Glossary Farm. Previously, the users had a separate account on each instance. The migration included a modification of the mapping to terms and definitions (creator, last editor). The migration was automated using a software. However, a relatively large number of transformations and jobs were created.

CHAPTER 10

Evaluation

This chapter presents the evaluation of the CIS farm prototype. First, the design of the evaluation is briefly explained, then the study process is described. Finally, the results are summarized and discussed.

10.1 Study Design

The study design is based on the guidelines provided by Robson [80], Runeson and Höst [81] and Wohlin et al. [82] for empirical research. Additionally, the *"General survey guidelines"* of Thiel [83] have been considered.

Robson [80] defines a survey as a

> *"[...] collection of standardized information from a specific population, or some sample from one, usually, but not necessarily by means of a questionnaire or interview"*

According to [81] the primary objective of a survey is portraying a situation or phenomenon. Therefore it is a descriptive methodology approach. The combination of qualitative and quantitative data provides a better understanding of the studied phenomenon. This approach for data collection is called "mixed methods". Quantitative data consist of numbers and classes, while qualitative data involves i.a. words and descriptions. The analysis method for quantitative data is statistics, whereas for the qualitative data categorization and sorting are used [81].

The research process may be subdivided into *fixed* and *flexible*. In a fixed design process, all parameters are pre-defined and remain unchanged after the launch of the study. In contrary, in a flexible design process key parameters can be adjusted during the study. Typically a survey has a fixed design process [81].

Robson [80] as well as Runeson and Höst [81] describe similar processes to perform a case study. The process is nearly the same for any kind of empirical study and the following five essential process steps should always be performed [81]:

1. **Design.** The objectives are set and the study is planned.
2. **Preparation for data collection.** Procedures and protocols for the data collection are specified.
3. **Collecting evidence.** Execution of the study with data collection on the studied case.
4. **Analysis of collected data.** The results are examined.
5. **Reporting.** The collected data are summarized and packaged.

This study is the foundation to answer research question 3, to what extent are the identified architectural principles and the farm evolution approach sufficient to describe key characteristics of a CIS farm. Therefore, the goal of the study is to figure out (1) how well the CIS farm approach is suitable for the glossary environment and (2) and how well the Glossary Farm supports researchers. The questions of the survey concern, in particular, the evaluation of the following points:

- Effort reduction for researchers
- Differentiation between glossary and farm components
- Linking of terms
- Locating relevant information
- Notifications
- Platform features

10.1.1 Participants

Since the Glossary Farm is primarily designed and developed for the scientific field, only persons with university background were contacted. To ensure that all participants have research reference, only employees, research contacts and students of the Institute of Information Systems Engineering were contacted. We aimed to guarantee that all participants understand the importance of uniform definitions of a term. We invited 50 persons via e-mail to be a participant in the survey.

10.1.2 Data Collection

According to Lethbridge et al. [84] we used a second degree technique for the data collection. The second degree technique is an indirect method, where the scientist collects raw data without interacting with the subject during the data collection. For the data collection we used a questionnaire, which was created with *Google Forms*[1] to ensure

[1] `https://www.google.com/forms/about`, last visited at 10.10.2018

that the participants could answer the questions online. The questionnaire consists of 21 questions with quantitative and qualitative answer options. Seven questions of the survey are of qualitative nature (open-ended questions), the remaining fourteen question are of quantitative nature (closed-ended questions). The questionnaire can be found in appendix A.2.

Figure 10.1 shows the performed action steps of the participants during the data collection. The following steps were executed by the participants:

1. **Read the E-Mail**
   Each participant received an invitation e-mail where the context of the survey was briefly explained. Furthermore, the attachment of the e-mail included a PDF file with test instructions and a detailed guide to use the Glossary Farm.
2. **Execute the Test Instructions**
   The test instructions included an *activity sequence*, which the participants should execute on the Glossary Farm to get familiar with the system. The *activity sequence* can be found in Appendix A.1.
3. **Evaluate the Glossary Farm Prototype**
   While the participants performed different activities, they collected various impressions and gained some experience with the glossary farm prototype.
4. **Complete the Survey**
   Finally, the participants answered the online questionnaire and completed the survey.

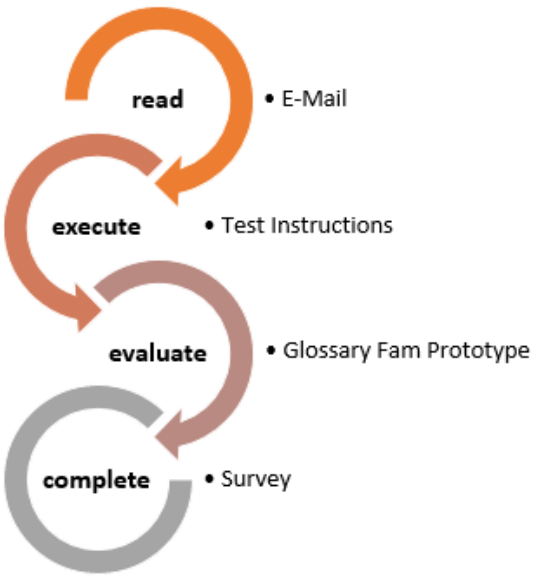

Figure 10.1: Action steps of the participants

10.1.3 Data Analysis

Runeson and Höst [81] mentioned that the data analysis is different for quantitative and qualitative data. For the analysis of quantitative questions we used descriptive statistics, such as mean values and histograms to get an understanding if the Glossary Farm fits the needs of the researchers. The hypothesis generation technique was used to analyze the qualitative questions. This technique is intended to find hypotheses from the data and the results are the hypotheses as such. For this, we went unbiased and open to the study in order to derive hypotheses from the data.

10.1.4 Threats to Validity

Wohlin et al. [85] denotes the validity of a study as

> *"[...] the trustworthiness of the results, and to what extent the results are true and not biased by the researchers' subjective point of view."*

According to Wohlin et al. [85] four types of validity threats exists: internal validity, external validity, conclusion validity and construct validity. In the following we describe some threats which are relevant in this study, referring to the terms and explanation mentioned of Wohlin et al. [85].

Internal Validity

The internal validity deals with whether an experimental treatment affects the outcome or not. Some factors with impact are how the subjects are selected and divided into different classes or if special events occur during the experiment. To avoid *instrumentation* threats from poorly designed artifacts, the questionnaire, the test instructions and the e-mail text were reviewed by experienced researchers. Only volunteers are selected for the questionnaire to prevent the *selection* threat, since volunteers are usually more motivated and appropriate for new assignments than the whole population. All participants used the Glossary Farm for the first time at the time of the survey to avoid the *maturation* effect, therefore all participants were unbiased.

External Validity

The external validity concerns the generalization of the results, such as how the findings are interesting to people outside of the investigated case. We avoid the effect *interaction of selection and treatment* by selecting participants with different roles and tasks in a research project. Furthermore, we asked the participants if they had any experience in creating a glossary to correlate the results with prior impressions. In addition, to avoid the *interaction of history and treatment* threat the questionnaire was available for several weeks and no changes were made to the Glossary Farm during that time.

Conclusion Validity

The conclusion validity deals with the relation between treatment and outcome of the experiment, there should be a statistical relationship. To avoid the *fishing and the error rate* threat, we examined the questions individually and unbiased. We did not fish for a specific result. To prevent the *random heterogeneity of subjects* effect we also querying the participants experience with a glossary. Therefore the risk that the variation of individuals is larger than due to the treatment is minimized.

Construct Validity

The construct validity is concerned with the relationship between theory and observation, whether the study represents what the researcher has in mind and what should be explored on the basis of research questions. To avoid the *inadequate preoperational explication of constructs* effect the constructs are sufficiently defined before they are transformed into measures or treatments. Furthermore, the quality of the results do not depend on a single quantitative metric - no *mono-operation bias* threat.

10.2 Results

This section summarizes the results of the survey. For the analysis of some questions, the participants were split into two groups: those who already had experience with the creation or management of a glossary and those who had none.

Participants

In the survey a total of 25 people, with different roles in the research sector participated (compare figure 10.2).

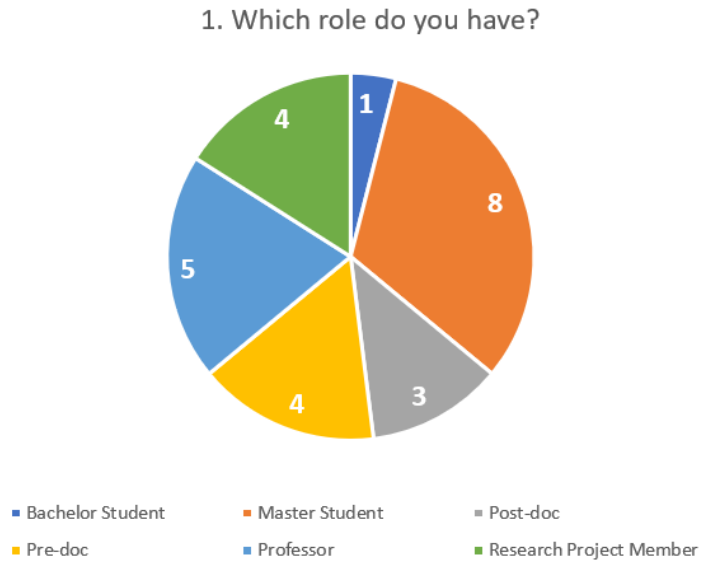

Figure 10.2: Roles of the participants

Effort reduction & benefits

If the participants are divided into experienced and inexperienced users, participants with experience have more often the impression that the effort can be reduced with the Glossary Farm (compare 10.3). Experienced participants also considered more often that the platform could be easily used for their purposes (compare 10.4). In addition, experienced participants noted more frequently that the Glossary Farm has extreme benefits for the involved stakeholders and supports the work of a researcher with project collaborators (compare 10.5).

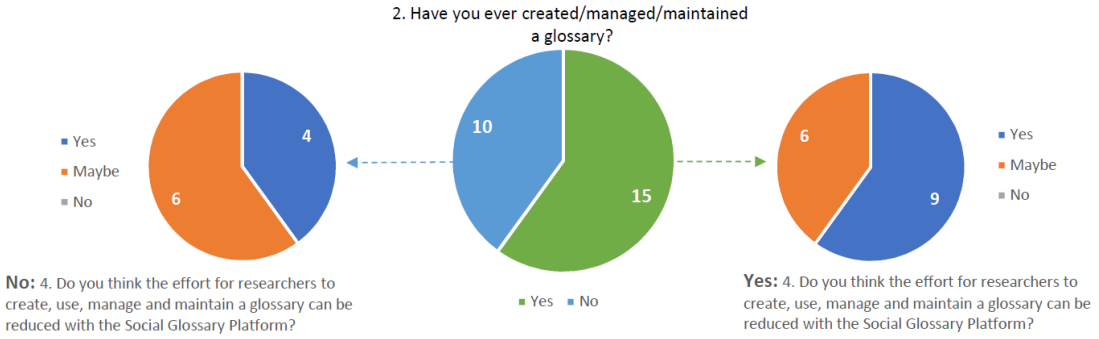

Figure 10.3: Effort reduction depending on the experience

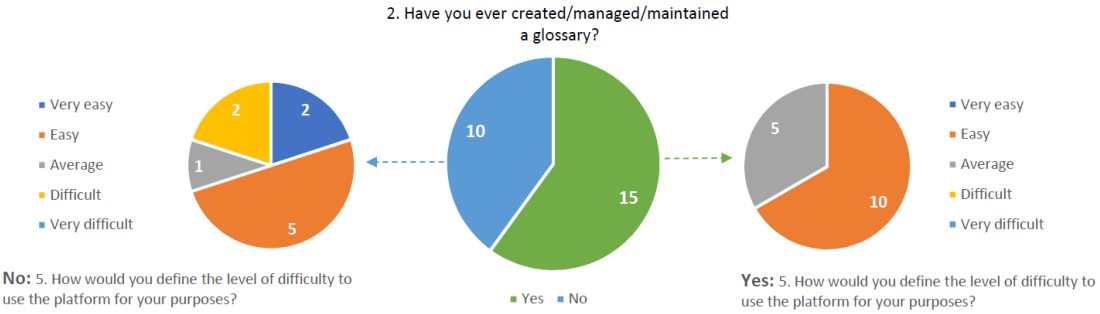

Figure 10.4: Level of difficulty to use the platform

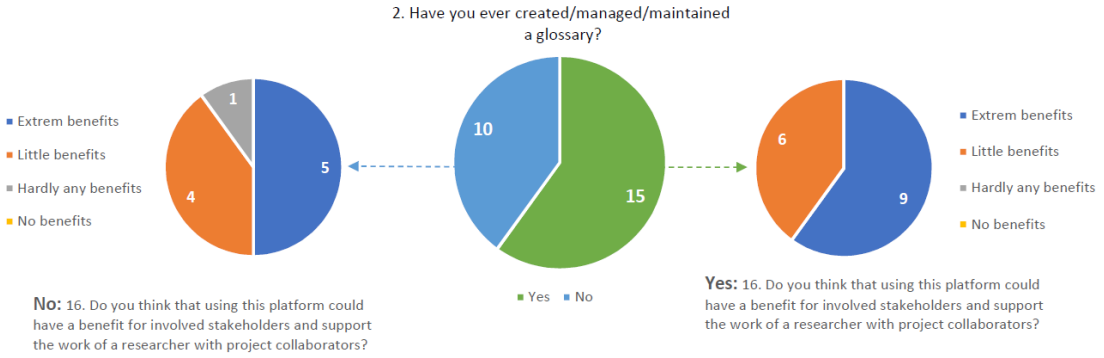

Figure 10.5: Benefits for stakeholders

Differentiation between glossary and farm components

96% (24 participants) think that the structure of different glossaries is useful. 76% of the participants had no difficulty to navigate between the glossaries. About half of the participants could easily distinguish between glossary and farm components, while 40% indicated the difficulty with *"average"*.

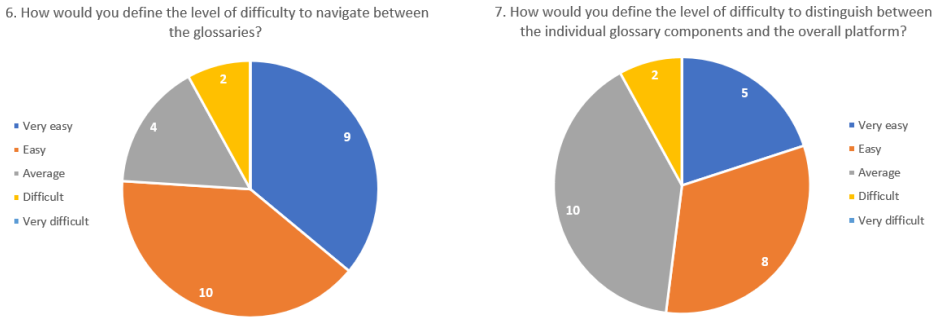

Figure 10.6: Level of difficulty to navigate and distinguish

Platform features

Another task of the participants was to evaluate several platform elements. Through the executed *activity sequence* the participants came in touch with all these elements. Most of the features were deemed to be useful. The ”glossary and term overview” were particularly well received. The elements ”tags and category overview”, ”discussion related to a term” and ”voting on definition” also obtained a good rating score. The usefulness of elements such as ”email notifications” or ”user profile sites” was instead seen as more average. The e-mail function probably scored an average because the users did not work with the system for a long period. They only have been working with the platform during the survey. Therefore, the participants did not receive any personal summaries, they only came into contact with the settings for notifications. The exact rating can be found in figure 10.7.

E-Mail Notifications

An important feature of the Glossary Farm are e-mail notifications. These are intended to inform users about content changes and activities of other users and should animate them to further contributions. The participants were surveyed on how often they want to receive an e-mail and whether they want an extra mail for each glossary. Most of them agreed that the default time frame is sufficient: once a week for the global news & personal digest and once a month for the ranking email. Additionally, several participants pointed out that too many e-mails could be annoying and the time frame should be adjustable. 84% prefer to receive an email notification with collected information on all glossaries where they are members. Some participants indicated that too many e-mails could quickly be considered as spam and because of this, they would instead prefer a summary.

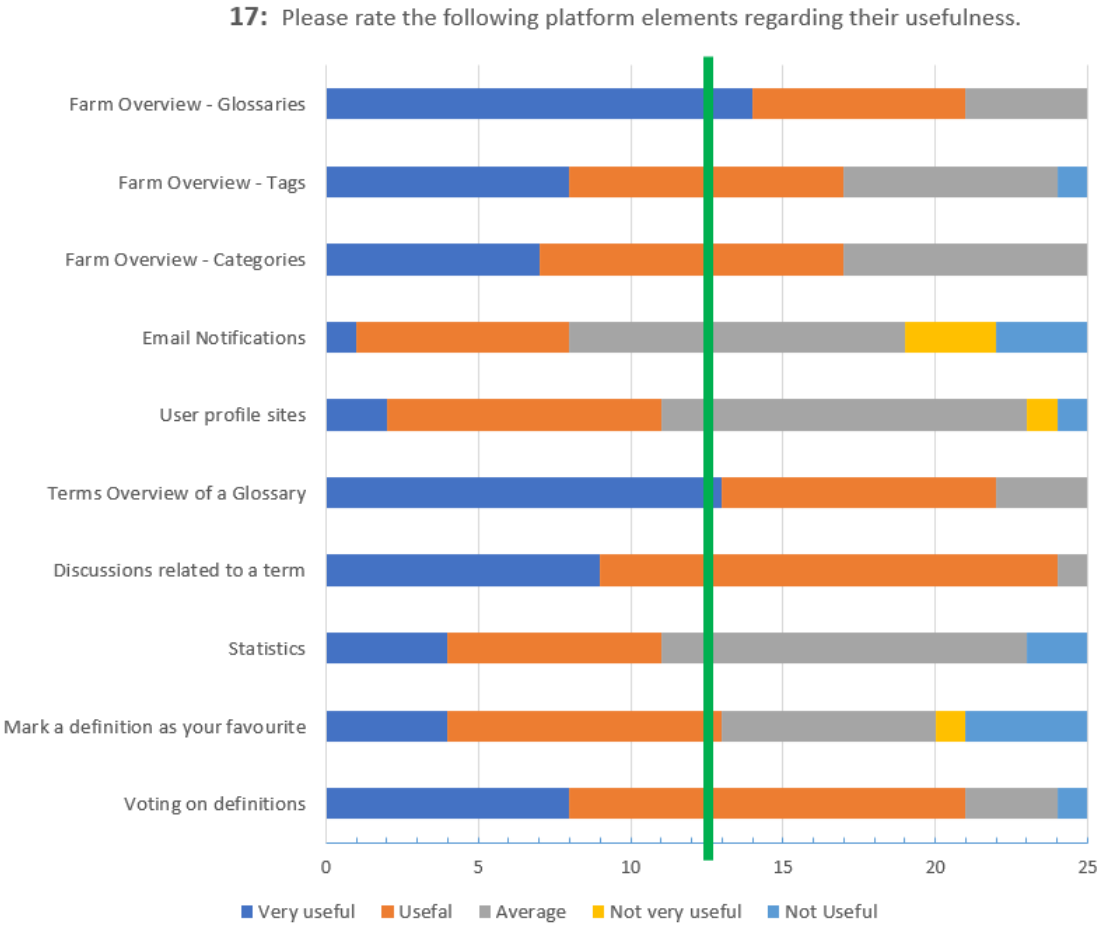

Figure 10.7: Rating of platform features

Link, Locate & Discover

Experienced participants found it easier to locate useful and relevant information in the Glossary Farm. Half of the inexperienced participants evaluated the discoverability with average or difficult, while experienced users rated both only with 26.67%. However, the majority of the participants found the discoverability as easy (compare figure 10.8). Most of the experienced and inexperienced participants agreed that the provided information by the platform and its artifacts is sufficient and useful for their purposes and goals (compare figure 10.9).

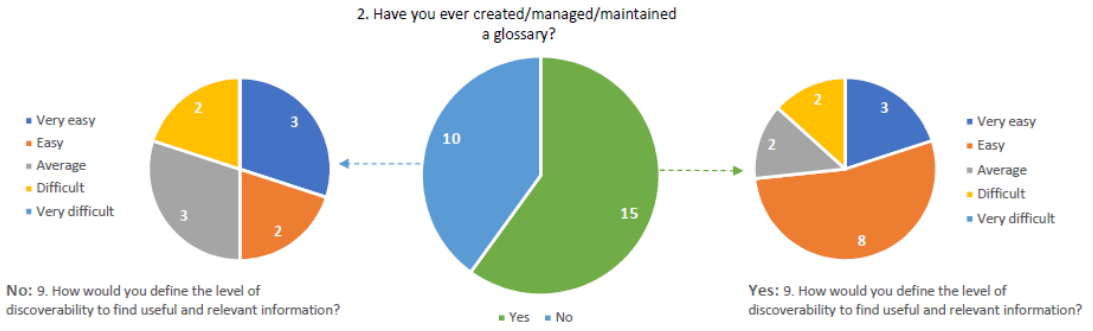

Figure 10.8: Level of discoverability

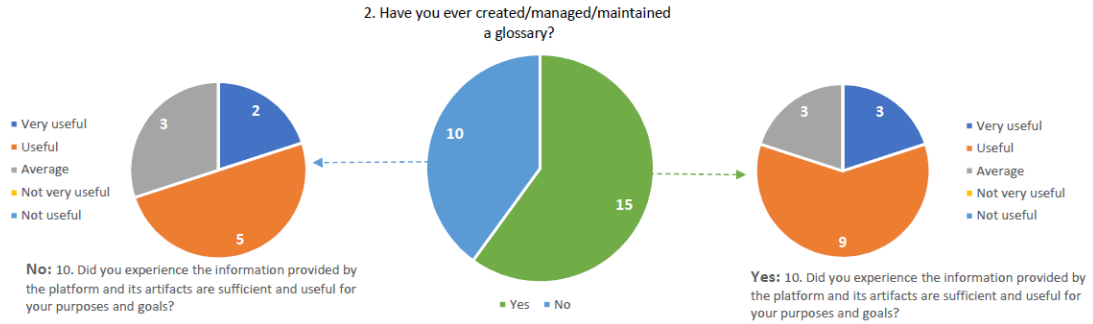

Figure 10.9: Usefulness of provided information

86% of the participants think that links between terms in different glossaries are useful. As advantages the participants specified the possibility (1) to see terms from different perspectives, (2) to switch easy and fast between various glossaries, (3) to connect different domains and (4) to exchange knowledge with another domain. Some participants pointed out a few minor issues, e.g., (1) it could be difficult to find related terms once there are many terms in the glossaries (2) the connection of terms could be better visualized (clarity can be lost).

The participants evaluated several statements regarding the Glossary Farm. Based on these statements, it can be concluded that the Glossary Farm provides a reasonable basis for the exchange of knowledge. Most agreed that the developed platform represents a trustful source where researchers can exchange terms and their various definitions. The links of terms across different glossaries provide a useful addition by extending the scope of the individual knowledge bases. The review workflow is a relevant and useful mechanism to ensure correctness and overall quality of a glossaries knowledge base. The classification of glossaries into categories and the collective use of tags enables an efficient structuring and a fast discovery of relevant knowledge. Figure 10.10 shows the rating for each individual statement.

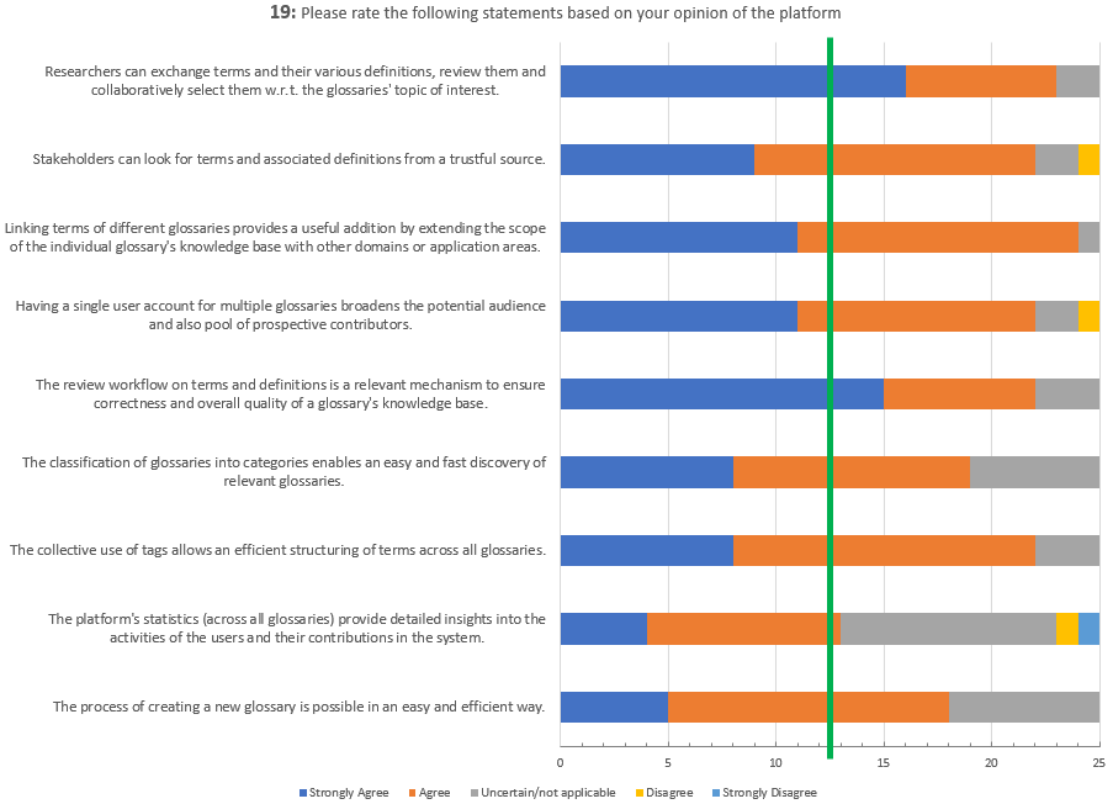

Figure 10.10: Rating of statements about the Glossary Farm

Suggestions for Improvements

The participants of the survey were also asked which features require further improvement to make them more useful. The following was pointed out:

- **Review Process.** Not every member of a glossary should be able to validate a term. This permission should explicitly be given only to some users, to ensure that the definition of a term is correct.
- **Rating stars.** A possibility should exist to uncheck a rating star again.
- **List of favorite definitions.** There should be an overview where all definitions marked as favorite are displayed.
- **Overview and search of User Profiles.** There should be a functionality to get an overview of all users and to search for users. Although this function already exists for administrators, it should be considered to enable this function for standard users.
- **Introduction.** A detailed description of the concrete process and the essential

functions of the platforms would allow users to become familiar with the system more quickly. This description should include, among others, (1) how terms can be created, (2) how terms can be linked, (3) how a glossary can be created, (4) what benefits tags and categories have and (5) what types of e-mail notifications exist.

- **FAQ.** An FAQ site should give an overview of all implemented features and a clarification who, where and what everybody can contribute to a glossary.

Additionally, the participants were also asked what functions they miss in the prototype. Suggestions for further features are:

- **Graph Visualization.** An appropriate visualization of connected terms would reveal better information about their relations.
- **Color-Coding.** Different colors for the glossaries to make the differentiation between them easier.
- **Badging system.** Some kind of badging system to motivate the users to contribute to a glossary.
- **Suggestions for synonyms/related terms.** Automated suggestions for synonyms and related terms to facilitate the search for possible connections.
- **Direct Messages.** Possibility to get in direct contact with other users to exchange relevant information faster.
- **Shortcuts.** Shortcuts for key activities (such as create term, edit term, etc.) to save time.

CHAPTER 11

Discussion

CIS may be useful with different kinds of knowledge base content, grouped and organized around a specific topic. Therefore, operators of a CIS platform want to reuse the system capabilities for knowledge with similar structures. This need leads to instance clones, where simply a copy of the CIS is filled with new knowledge and evolves more or less independently from the original system. However, through the independent deployment, various limitations arise, such as (1) a large number of instances, (2) many accounts for one researcher or (3) no cross-instance linking of knowledge possible. So a main disadvantage of the single deployment strategy is the high administration effort, the operators must maintain a high number of instances and the users posses for every instance where they are members a separate account. Furthermore, updates and bug fixes have to be executed separately for each system.

To eliminate all mentioned limitations this work proposed a new architecture approach, the CIS farm architecture, which is a particular variant of a system of systems architecture. Several CIS instances are integrated into one system environment and share specific functions. In the context of a CIS farm, a CIS instance is called a farm entity. A CIS farm provides functions to create, deploy and administrate individual and independent farm entities.

Only with a CIS farm architecture it is possible to:

- host several farm entities on a single, shared platform
- create and maintain a farm instance with a minimum on effort
- link knowledge across various instances
- single user account for contributions in several farm entities
- create notifications based on activities from various farm entities
- generate statistics and analysis over multiple farm entities

A different possible approach that we initially considered was container-based virtualization. With this approach, a container encloses a CIS and isolates the application. Therefore, updates or bug fixes in the application can be automated via routines. New containers (CIS instances) can be easily created with the use of a template. Therefore, through virtualization, the cost of maintenance can be dramatically reduced. However, some limitations such as single user account or integrated data analysis could not be resolved with container-based virtualization. Another different approach which was considered was all-in-one. Thereby the whole knowledge is collected in one CIS instance. With this approach, the administration effort can be reduced. Besides, users have only one account, and the data analysis is possible in greater detail. However, new emerging problems and limitations arise, such as (1) no differentiation of communities is possible and (2) the loss of clarity between different domains and content. So, we think the CIS farm approach is the best solutions to integrate several CIS instances in one system environment and to share knowledge between them.

RQ 1: What are the underlying architectural principles of a CIS farm platform?

To answer RQ 1, we identified several architecture principles and evolved the CIS architecture meta-model of Musil et al. [2] to describe a CIS farm architecture. Furthermore, we have applied the stigmergic process of Musil et al. [2] to the CIS farm to ensure that a feedback loop exists, which motivates human actors to further contributions. The stigmergic process of a CIS farm consists only of one additional element that takes care of the execution of the dissemination rules within the farm entities. Therefore, the CIS farm uses the same operating procedure as a CIS and applies an information-gathering model that is well-defined and successful.

The identified CIS farm architecture principles can be found in several platforms in practice. However, it should be noted that depending on the goal of the platform, the design decisions how to apply the architectural principles in a CIS farm architecture can differ. For example, the philosophy of Wikia[1] is that they support only open and collaborative community projects. Therefore, it is not possible to change the visibility of wiki projects (farm entities) or to restrict the users who can contribute. In contrast, ShoutWiki[2], also a wiki farm, follows a completely different philosophy. Within ShoutWiki it is possible to limit the visibility of the wikis and make them accessible only to specific users. However, ShoutWiki limits overlapping search across wikis, searching is only possible in a wiki. Table 11.1 gives an overview of differences in the design of a CIS farm architecture using the examples of Wikia and ShoutWiki.

[1] `https://www.fandom.com`, last visited at 10.03.2019
[2] `http://www.shoutwiki.com`, last visited at 10.03.2019

Principle	**Wikia**	**ShoutWiki**	**Glossary Farm**
CIS farm administrator	✓	✓	✓
Global unique user account	✓	✓	✓
Global individual user profile	✓	✓	✓
System-wide permissions	✓	✓	✓
Process to create new farm entity (users can create a farm entity)	✓	✓	✓
Classification of farm entities in categories	✓	✓	✓
Visibility of the farm entities	Everything is public	Different settings (private or public access possible)	Different settings (private or public access possible)
User assignment to farm entities	Every registered user can contribute	Depending on the visibility setting	Only members can contribute to a farm entity
Individual permission for every farm entity	✓	✓	✓
Artifacts belong to a farm entity	✓	✓	✓
Create links between different farm entities	✓	✓	✓
Artifact search	Depends on the view (whole platform or just in the farm entity)	Only per wiki	Depends on the view (whole platform or just in the farm entity)
Notification system	✓	✓	✓
Individual notification per farm entity	✓	✓	✓
Notification setting for each farm entity	✓	✓	✓

Table 11.1: Differences in CIS farm architecture design decisions in practice

As shown in table 11.1 all mentioned CIS farms have implemented the architecture principles in one way or another. However, it depends on the platform and its requirements on how the architecture principles are realized. For example, in Wikia all farm entities are public, but ShoutWiki or the Glossary Farm have different visibility settings. Therefore it is a design decision how a CIS farm implements the architecture principles.

In summary, the findings of RQ 1 help researches and software architects to better understand the principles and design a CIS farm architecture. A CIS farm collects knowledge around a specific topic in farm entities and shares this knowledge between multiple human actors. Thereby the goal is to increase the knowledge content of the farm entities through constant contributions of the human actors.

RQ 2: What are the steps to evolve a CIS into a farm platform?

To answer RQ 2, we designed an evolution process that assists software architects with a step-by-step guide to evolve a CIS to a CIS farm. During the literature study it turned out that no suitable approach for the evolution of a CIS to a CIS farm could be found. Only general statements that facilitate software evolution could be identified. The challenge in evolving a CIS system to a CIS farm is to build an enclosing system that includes several CIS instances, so-called farm entities. In addition, a CIS farm requires additional features, such as a user and farm entity administration function. All these features must be considered and designed.

To answer RQ 2 satisfactorily, we designed the CIS-EVO-Farm approach, a light-weight, decision-tree-based process that supports the architectural evolution of a CIS platform into a CIS farm. This approach assists developers and software architects with a step-by-step guide to evolve a CIS platform into a CIS farm. In order to apply the CIS-EVO-Farm approach successfully, the following criteria must be fulfilled: (1) access to the CIS code base, (2) access to the CIS data and (3) all farm entities must have the same structure.

We applied the CIS-EVO-Farm approach to evolve the Glossary Platform into a Glossary Farm. Steps 1 - 3 of the evolution process were easy and quick to implement. These steps have mainly concerned the application and features of the Glossary Farm. During step 4 the architecture design was evolved to fulfill the requirements of a CIS farm. In step 5, the database schema was adjusted so that data can be stored to match the CIS farm structure. In step 6 the Glossary Farm was implemented, which was very time consuming. After thoroughly testing the Glossary Farm in step 7, we concluded that the designed platform operates satisfactorily. Particular attention was paid to testing the aggregation and dissemination phase so that the knowledge is correctly collected and distributed to the users. Finally, in step 8, existing data have been migrated to the Glossary Farm. This step was also time consuming since various database tables have changed and no one-to-one transmission to the Glossary Farm was possible. Furthermore, the *IDs* of the data had to be changed so that a correct mapping in the new database schema is possible.

Through the practical use of the CIS-EVO-Farm approach, we were able to verify the approach and adapt it if necessary. However, an adaptation was not necessary because the application of the evolution process was easily possible. We think that the evolution process is a well-defined guide for software architects and developers to evolve a CIS to a CIS farm.

In summary, the findings of RQ 2 support software architects and developers in the evolution of a CIS to a CIS farm. Our designed approach, the CIS-EVO-Farm approach, is only valid for the evolution of a CIS into a CIS system-of-system farm platform.

RQ 3: To what extent are the identified architectural principles and the farm evolution approach sufficient to describe key characteristics of a CIS farm?

To answer RQ 3, we implemented the Glossary Farm where we applied the findings of RQ 1 and RQ 2 to verify them. After the implementation, we conducted a user study to evaluate the Glossary Farm. Most of the participants said that the Glossary Farm can reduce the effort of creating, using and maintaining a glossary. Participants with previous experience in the creation and administration of glossaries have seen more benefits in the CIS farm approach. These participants rated questions about the simplicity and usefulness of the application better. The conclusion of the study is that the Glossary Farm provides a reasonable basis for the exchange of knowledge. Most participants agreed that the application represents a trustful source where researchers can exchange terms and their various definitions. Furthermore, the links of terms across different glossaries provide a useful addition by extending the scope of the individual knowledge bases.

The results of the user study show positive reactions and the participants were enthusiastic about the CIS farm approach that integrates several CIS instances into one system environment. However, only a minimal set of features were implemented to keep the effort manageable during the implementation. This opens a broad scope for suggestions for improving the applicability and usefulness of the Glossary Farm. Therefore, the participants provided a couple of suggestions for new features, e.g., the graph visualization of connected terms or automated recommendations for synonyms and related terms. We are sure if some of these proposed features are implemented in the future, the usefulness of the Glossary Farm would increase.

In summary, by applying (1) the CIS farm architecture principles, (2) the CIS farm meta-model and (3) the CIS-EVO-Farm approach a software architect is able to design and evolve a CIS to a CIS farm.

Similarities and Differences to Related Work

To the best of our knowledge no architecture approach exist in the literature, where several CIS instances are embedded in one environment and share functionalities. Therefore, we searched for architecture principles, related approaches and evolution process to gather ideas for the design of a CIS farm architecture.

Garlan [11] highlighted the importance of a good architecture as it can increase the overall quality of the application. Therefore, we provided several architecture principles and a meta-model to support software architects in developing a CIS farm. During the architecture modeling we considered a variety of design principles to maximize the usability and extendibility [19, 20].

Vergados et al. [28] described several requirements that a collective intelligence application should fulfill. Most of these requirements are considered in the architecture description of the CIS farm. Since the CIS farm provides no web service interfaces to facilitate the re-use of data the requirement *"facilitate data access"* is violated.

We extended the CIS architecture meta-model of Musil et al. [2] to fulfill the needs of a CIS farm. Furthermore, we used the stigmergic process of this work to describe the feedback loop of a CIS farm. We added only one additional element to take care of the execution of the dissemination rules within the farm entities. The principle of the stigmergic process is the same between CIS and CIS farm.

Guessi et al. [8] mentioned the five main characteristics of an SoS approaches. Although the CIS farm and SoS have some similarities, some of these characteristics are violated. For example, all farm entities provide exactly the same functions, they only manage different data. However, in an SoS, the subsystems have different functionalities and can exist independently.

Within a multi-tenant environment, tenants share resources, but have their own delimited area and can customize the application individually to their needs [45]. This is a main contradiction to the CIS farm approach since all member of a CIS farm share their data and can contribute to several farm entities.

The database design for multi-tenant systems is a widely discussed issue, i.a. by Chong et al. [50], Karatas et al. [51] and Wang et al. [52], since there exist varying degrees of data isolation. For the database design of the Glossary Farm, we used the totally shared principle (shared database, shared schema) since the data of all farm entities can be linked.

The world is continuously changing, new business opportunities occur over time. Therefore there is a need to adjust the software. The largest part of life cycle costs flows into the evolution of software to respond to the changing requirements [5]. This makes it especially important to design architecture approaches that are easily extendable. Accordingly, we design the CIS farm approach, where easily new CIS farm entities can be created. By applying this approach, it is possible to provide new functions for many individual communities without updating each CIS instance individually.

In the literature, no suitable evolution process could be identified that supports the evolution of a CIS to a CIS farm. Only general statements that facilitate software evolution could be identified. For example, an object-oriented or component-based design improves the reusability [64, 79].

According to Aoyama [68] the evolution of a CIS to a CIS farm is basically a discontinuous software evolution since some essential aspects such as software architecture and features are changed.

Limitations of this thesis

There are some limitations of this thesis. In the prototype, only the essential functions are implemented to verify the CIS farm approach. During the user study, the participants proposed some useful features that would increase the applicability of the Glossary Farm, such as different color coding for the glossaries or graph visualization of the connected terms. Furthermore, we put a limited focus on the usability of the Glossary Farm to implement the concept of a CIS farm. This was then partially criticized in the user study, e.g., missing introduction or help page.

The architecture meta-model of Musil et al. [2] has been extended to allow multiple CIS instances to exist in one environment. No much implementation effort was attached to security, privacy and data protection. Besides, our architecture model does not support the interoperability between applications as well as other CIS farms. There are no web services interfaces available which allow data exchange with other applications.

Furthermore, we applied the new architecture approach only in one case, the Glossary. It should be applied in other application scenarios to strengthen the outcomes.

CHAPTER 12

Conclusion

There is an ongoing interest to optimize the process of knowledge creation and sharing. A wide range of researchers are engaged in this topic. In the last few years, web-based social platforms have been developed that use the collective intelligence of connected groups of people. These platforms efficiently collect and distribute their content among their user base and can be referred as Collective Intelligence Systems (CIS).

The Glossary Platform is a CIS, which helps researches from multiple domains to collaborate geographically independent by using a shared online platform. In a glossary scientists collect and share terms and their various definitions, either for their domain of study or project-specific knowledge. The existing glossary solution of the TU Wien has several restrictions, e.g., for every research collaboration a separate glossary instance needs to be created and deployed. From these restrictions, again new limitations are derived, for example (1) a large number of instances, (2) many accounts for one researcher and (3) no cross-instance linking of knowledge possible. Therefore, the Glossary Platform should be evolved.

To eliminate all mentioned limitations a new architecture approach has been designed, the CIS farm architecture. A CIS farm provides functions to simply create, deploy and administrate individual and independent farm entities. A farm entity is a CIS instance, which is embedded in the farm environment and shares several functions with other farm entities. The focus of the thesis was to define this CIS farm approach and to delimit it from other similar approaches.

As a first step, existing literature was investigated to identify similar architecture concepts. We could identify two similar architecture approaches, the system of system architecture and the multi-tenant architecture. However, both approaches pursue goals too different as to be used as a foundation for a CIS farm architecture. An essential difference to SoS is the CIS farms violation of the principles *geographical distribution* and *managerial independence*. Farm entities are embedded in a single environment and cannot exist alone.

Furthermore, the farm entities have no independent life cycle, they are bound to the life cycle of the global system. If the global system crashes or shuts down, the farm entities are also no longer reachable. In contrast to multi-tenant architecture, the farm entities are *global available* to all users and not isolated and invisible to others. It is, therefore, possible to *link artifacts* of different farm entities.

Additionally, we reviewed available systems with a CIS farm approach in chapter 4 to identify and categorize common features and capabilities. One challenge was to find similarities in widely differing systems in diverse application areas. Nevertheless, several common features could be identified, such as (1) single user accounts, (2) a simple creation procedure of a new farm entity and (3) regular notifications about recent activities in the system.

Based on the results of the literature review and the survey about existing CIS farm platforms the architecture principles of a CIS farm were derived. Examples for this principles are *a single user account* or *links between different farm entities*. Based on the SIS architecture pattern of [2] the CIS farm meta-model was defined. This meta-model contains all necessary components and principles to design a CIS farm architecture. In addition, the stigmergic process of Musil et al. [2] was described in context of the CIS farm architecture. The stigmergic process is a permanent feedback loop, which should motivate human actors to further contributions. To fulfill the requirements for a stigmergic process on farm level, we added a new element, the dissemination routine. The dissemination routine in each farm entity deals only with the execution of the dissemination rules, which are defined at the global level. Therefore, the stigmergic process of CIS is equal to that of a CIS farm, it supports only various farm entities.

To support software architects and developers in the evolution of a CIS platform to a CIS farm we introduced the CIS-EVO-Farm approach. The CIS-EVO-Farm approach is a step-by-step guide for the architecture evolution. In order to apply the CIS-EVO-Farm approach successfully, the following criteria must be fulfilled: (1) access to the CIS code base, (2) access to the CIS data and (3) all farm entities must have the same structure. The CIS-EVO-Farm approach consists of the following steps:

1. Identification of CIS farm and farm entity properties
2. Characterize user groups
3. Determine the permission system
4. Evolve the system architecture
5. Design database schema
6. Implementation of the new structure and features
7. Verification of the CIS farm
8. Migrate Data to the CIS farm

After the completion of all steps, a CIS farm was successfully implemented. In addition, we developed a guide for the data migration process to transfer the data from various established CIS platforms to the CIS farm environment.

To verify our contributions, we implemented the Glossary Farm where we applied the identified architecture principles for a CIS farm. Furthermore, we used the CIS-EVO-Farm approach to evolve the Glossary Platform to a Glossary Farm. Afterwards, we evaluated the Glossary Farm using a user study. The user study was divided into several steps. First, the participants had to read the received test instructions and execute them afterwards in the Glossary Farm. Thereby, the participants became familiar with the application and were able to complete the questionnaire based on the received impressions. Most of the participants said that the Glossary Farm can reduce the effort of creating, using and maintaining a glossary. Participants with previous experience in the creation and administration of glossaries have seen more benefits in the CIS farm approach. These participants rated questions about the simplicity and usefulness of the application better. The conclusion of the study is that the Glossary Farm provides a reasonable basis for the exchange of knowledge. Most participants agreed that the application represents a trustful source where researchers can exchange terms and their various definitions. Furthermore, the links of terms across different glossaries provide a useful addition by extending the scope of the individual knowledge bases.

In summary, the contributions of this thesis provide a solid foundation for researches, software architects and developers to describe and design a complex system-of-systems CIS. We defined the CIS farm architecture approach to describe how several CIS instances can be embedded into one system environment. Through this approach, the administration effort can be reduced dramatically and allows an integrated data analysis and the exchange of data through several CIS instances, so-called farm entities. Through the CIS farm meta-model and the CIS-EVO-farm approach, we provide a method and guidance to design a CIS farm architecture and to support the architecture evolution of a CIS to a CIS farm.

There are a lot of possibilities for continuing the research in this subject area. The introduced architectural meta-model provides a solid foundation, but also offers range for enhancement. It could be interesting to categorize CIS farms into different subtypes and to identify their differences and what these differences mean for their architecture design. Furthermore, it would be conceivable to develop an architecture viewpoint, framework or reference architecture for CIS farms.

Besides, it is possible to extend the feature set of the Glossary Farm. These features could be considered as an optional extension set, built-in as needed into the concrete architecture of the Glossary Farm. In the following, some valuable features for a Glossary Farm are described:

- **Extend functions for farm entities**
  Farm entities could have an extended set of features. Some features are only available for certain farm entities. An administrator could unlock these features for a farm entity, such as a chat system for a farm entity.
- **Interface to exchange data with other CIS farms.**
  A functionality could be added, where different CIS farm exchanges their knowledge automatically. This could be in the form of links pointing to relevant content in another CIS farm.
- **Machine learning algorithm for CI artifact recommendation.**
  A machine learning algorithm can be implemented to give users suggestions for interesting CI artifacts in the dissemination phase. Over time the algorithm suggests more appropriate CI artifacts, e.g., the algorithm learns based on the user clicks on CI artifacts of e-mails.

In addition, to increase the usefulness of the Glossary Farm, suggested features of the participants of the user study could be implemented. Some of these features are very expensive to implement but provide the users better applicability and usability. Some new Glossary Farm features could be:

- A graphical visualization which provides a better overview of connected terms and their relations.
- A badging system which motivates the users for further and more contributions.
- An automated suggestion system which recommends synonyms and related terms for a term.

Moreover, it would be conceivable and beneficial to evaluate the CIS farm architecture approach in another application scenario which could strengthen the outcomes of this work. This could be of high importance, since in the modern world the interest of networked and collectively created knowledge bases is growing rapidly.

APPENDIX A

Survey

A.1 Activity sequence of the test instructions

The invitation e-mail contained a PDF file with test instruction, which included a sequence of activities. This activity sequence had to be performed by the participants to get familiar with the Glossary Farm. The content of the PDF file is shown below.

Instructions
This survey investigates the applicability and usefulness of the Glossary Platform prototype, which aims to support the collaborations among researchers and their partners by facilitating the creation and management of several glossaries. A typical use of a glossary in research collaborations is to reduce terminological misunderstandings and inconsistencies by providing a single agreed definition for a particular term. The Glossary Platform aims to minimize the administration effort, which arises, e.g., from handwritten notes or Word documents, including editing, sharing, access, track keeping, monitoring, distributing, and commenting.

On the Glossary Platform you can find several online glossaries, each shared between members of different user groups who have a central access to the collected knowledge about terms. A glossary allows researchers from multiple domains to collaborate geographically independently and to collect and share terms and their various definitions in a field/domain

1. Perform Activity Sequence on Prototype

1. **Open** the Glossary platform prototype: `http://glossary-farm-dev.herokuapp.com`
2. **Login** (username: survey, password: survey).
3. **Open** the **Glossary** "Survey".

4. **Create** a new **term** with at least one **definition** (feel free to choose a term with which you are familiar).
5. **Relate** your new term with the term ”Example” of the Glossary ”CIS-Farm” either by using the ”Synonym” or ”Related Term” relation.
6. **Vote** on a recently created definition of a term.
7. **Mark** a definition as your **favourite**.
8. **Open** the “Tag overview” of the Glossary Platform and click on the tag ”CI”.
9. **Add** a new **comment** to the term ”Example” in the Glossary ”CIS-Farm”.
10. **Open** the ”Communication Activity Settings” under your ”Settings”.
11. **View** the **user profile** of the user ”Farm Admin”.
12. **Open** the **category overview** and **click** on the **category** ”Software Engineering”.
13. **View** the **statistics** of the Glossary Platform.

2. Answer Questionnaire
If you are ready, please fill in your answers to the questionnaire: `https://goo.gl/forms/YDOqcQ7Lh5uYqKol2`

Answering the questions should not take longer than 10-15 minutes. All your contributions and answers will remain confidential and will be used only for evaluation purposes. After completing the study, all provided information will be deleted.

We highly appreciate your feedback.

A.2 Structure of the Questionnaire

The structure of the questionnaire can be considered below. All questions are listed with all the possible answer options.

#	**Question text**	**Answer options**
1	Which role do you have?	◦ Professor ◦ Post-doc ◦ Pre-doc ◦ Master Student ◦ Bachelor Student ◦ Research Project Member
2	Have you ever created / managed / maintained a glossary?	◦ Yes ◦ No
3	If yes, what form was chosen for the glossary?	◦ a hand-written document ◦ an editor-based document like Word or Latex document ◦ a table-based document like a spreadsheet ◦ online document ◦ wiki ◦ others
4	Do you think the effort for researchers to create, use, manage and maintain a glossary can be reduced with the Social Glossary Platform?	◦ Yes ◦ Maybe ◦ No
5	How would you define the level of difficulty to use the platform for your purposes?	◦ Very easy ◦ Easy ◦ Average ◦ Difficult ◦ Very difficult

6	How would you define the level of difficulty to navigate between the glossaries?	◦ Very easy ◦ Easy ◦ Average ◦ Difficult ◦ Very difficult
7	How would you define the level of difficulty to distinguish between the individual glossary components and the overall platform?	◦ Very easy ◦ Easy ◦ Average ◦ Difficult ◦ Very difficult
8	Do you think the structure of the different glossaries is useful?	◦ Yes ◦ No
9	How would you define the level of discoverability to find useful and relevant information?	◦ Very easy ◦ Easy ◦ Average ◦ Difficult ◦ Very difficult
10	Did you experience the information provided by the platform and its artifacts are sufficient and useful for your purposes and goals?	◦ Very useful ◦ Useful ◦ Average ◦ Not useful ◦ Not very useful
11	Is it easy to participate in the process, in discussions or to create new contributions?	◦ Yes, everything worked fine. ◦ No, it did not work for me.
12	If not, what were occurring problems?	*free text*
13	Do you think the links between terms in different glossaries is useful (cross-reference knowledge)?	*free text*

14	Currently, by default you receive a global news & personal digest once a week, and a ranking email showing all involved glossaries once a month. Do you think receiving this structure of email notifications and time frame is useful?	*free text*
15	Would you prefer to receive email notifications for each involved glossary? If not, why?	*free text*
16	Do you think that using this platform could have a benefit for involved stakeholders and support the work of a researcher with project collaborators?	◦ Extreme benefits ◦ Little benefits ◦ Hardly any benefits ◦ No benefits
17	Please rate the following platform elements regarding their usefulness.*(matrix)* ◦ Farm Overview - Glossaries ◦ Farm Overview - Tags ◦ Farm Overview - Categories ◦ Email Notifications ◦ User profile sites ◦ Terms Overview of a Glossary ◦ Discussions related to a term ◦ Statistics ◦ Mark a definition as your favourite ◦ Voting on definitions	◦ Very useful ◦ Useful ◦ Average ◦ Not useful ◦ Not very useful
18	Why do you think these elements are useful / not useful?	*free text*

19	Please rate the following statements based on your opinion of the platform.*(matrix)* ◦ Researchers can exchange terms and their various definitions, review them and collaboratively select them w.r.t. the glossaries' topic of interest. ◦ Stakeholders can look for terms and associated definitions from a trustful source. ◦ Linking terms of different glossaries provides a useful addition by extending the scope of the individual glossary's knowledge base with other domains or application areas. ◦ Having a single user account for multiple glossaries broadens the potential audience and also pool of prospective contributors. ◦ The review workflow on terms and definitions is a relevant mechanism to ensure correctness and overall quality of a glossary's knowledge base. ◦ The classification of glossaries into categories enables an easy and fast discovery of relevant glossaries. ◦ The collective use of tags allows an efficient structuring of terms across all glossaries. ◦ The platform's statistics (across all glossaries) provide detailed insights into the activities of the users and their contributions in the system. ◦ The process of creating a new glossary is possible in an easy and efficient way.	◦ Strongly Agree ◦ Agree ◦ Uncertain/not applicable ◦ Disagree ◦ Strongly Disagree
20	Which features require further improvement to make them more useful?	*free text*
21	What functionalities do you miss in the prototype?	*free text*

Table A.1: Questions of the Survey

Acronyms

AMD Analysis, Management and Dissemination System.
AR Actor Record.

CI Collective Intelligence.
CIS Collective Intelligence Systems.
CIS-AF Architecture Framework for Collective Intelligence Systems.
CRM Customer Relationship Management.
CS Constituent System.
CSCW Computer-Supported Cooperative Work.

Q&A Questions and Answers.
QoS Quality of Service.

RQ Research Question.

SaaS Software as a Service.
SIS Stigmergic Information System.
SLA Service Level Agreement.
SOA Service-Oriented Architecture.
SoS Systems of Systems.
SoSE System of Systems Engineering.

URL Uniform Resource Locator.

VM Virtual machine.

List of Figures

List of Tables

Bibliography

[1] Daniele Miorandi, Vincenzo Maltese, Michael Rovatsos, Anton Nijholt, and James Stewart, editors. *Social Collective Intelligence: Combining the Powers of Humans and Machines to Build a Smarter Society*. Springer International Publishing, 2014.

[2] Juergen Musil, Angelika Musil, and Stefan Biffl. SIS: An Architecture Pattern for Collective Intelligence Systems. In *Proceedings of the 20th European Conference on Pattern Languages of Programs - EuroPLoP '15*, pages 1–12. ACM, 2015.

[3] Toby Segaran. *Programming Collective Intelligence*. O'Reilly Media, first edit edition, 2007.

[4] Juergen Musil, Angelika Musil, and Stefan Biffl. Introduction and Challenges of Environment Architectures for Collective Intelligence Systems. *Agent Environments for Multi-Agent Systems IV*, pages 76–94, 2015.

[5] Hongyu Pei Breivold, Ivica Crnkovic, and Magnus Larsson. A systematic review of software architecture evolution research. *Information and Software Technology*, 54 (1):16–40, 2012.

[6] Teemu Kämäräinen, Yuanqi Shan, Matti Siekkinen, and Antti Ylä-Jääski. Virtual Machines vs. Containers in Cloud Gaming Systems. In *Proceedings of the 2015 International Workshop on Network and Systems Support for Games*, NetGames '15, pages 1–6. IEEE, 2015.

[7] John Paul Walters, Vipin Chaudhary, Minsuk Cha, Salvatore Guercio Jr., and Steve Gallo. A Comparison of Virtualization Technologies for HPC. In *22nd International Conference on Advanced Information Networking and Applications (aina 2008)*, pages 861–868. IEEE, 2008.

[8] Milena Guessi, Valdemar V. G. Neto, Thiago Bianchi, Katia R. Felizardo, Flavio Oquendo, and Elisa Y. Nakagawa. A systematic literature review on the description of software architectures for systems of systems. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing - SAC '15*, pages 1433–1440. ACM, 2015.

[9] Jaap Kabbedijk, Cor-Paul Bezemer, Slinger Jansen, and Andy Zaidman. Defining multi-tenancy: A systematic mapping study on the academic and the industrial perspective. *Journal of Systems and Software*, 100:139–148, 2015.

[10] Project: WikiFarm, 2017. URL https://www.mediawiki.org/wiki/Project:WikiFarm. Last visited at 14.10.2018.

[11] David Garlan. Software architecture: a roadmap. In *Proceedings of the conference on The future of Software engineering - ICSE '00*, pages 91–101. ACM, 2000.

[12] Len Bass, Paul Clements, and Rick Kazman. What Is Software Architecture? In *Software architecture in practice*, chapter 1. Addison Wesley, third edition, 2012.

[13] Mary Shaw and David. Garlan. Introduction. In *Software Architecture: Pespectives on an Emerging Discipline*, chapter 1, pages 1–18. Prentice Hall, 1996.

[14] Richard N. Taylor, Nenad. Medvidovic, and Eric M. Dashofy. Architectures in Context: The Reorientation of Software Engineering. In *Software Architecture: Foundations, Theory, and Practice*, chapter 2, pages 23–55. Wiley, 2010.

[15] Kai. Qian, Xiang. Fu, LiXin. Tao, Chong-wei. Xu, and Jorge. Diaz-Herrera. Introduction to Software Architecture. In *Software Architecture and Design Illuminated*, chapter 1. Jones and Bartlett Publishers, 2010.

[16] M. Galster, A. Eberlein, and M. Moussavi. Systematic selection of software architecture styles. *IET Software*, 4(5):349–360, 2010.

[17] A. Bijlsma, B.J. Heeren, E.E. Roubtsova, and S. Stuurman. Introduction to software architecture. In *Software architecture*, pages 6–17. FTA 2011 Free Technology Academy, 2011.

[18] Paul. Clements, Felix. Bachmann, Len. Bass, David. Garlan, James. Ivers, Reed. Little, Robert. Nord, and Judith Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, 2003.

[19] patterns & practices Developer Center. Chapter 2: Key Principles of Software Architecture, 2009. URL https://msdn.microsoft.com/en-us/library/ee658124.aspx. Last visited at 04.07.2018.

[20] Tutorialspoint. Software Architecture and Design Key Principles. URL https://www.tutorialspoint.com/software_architecture_design/key_principles.htm. Last visited at 04.07.2018.

[21] Ioanna Lykourentzou, Dimitrios J. Vergados, and Vassili Loumos. Collective intelligence system engineering. In *Proceedings of the International Conference on Management of Emergent Digital EcoSystems - MEDES '09*, page 134. ACM, 2009.

[22] Pierre Lévy. From social computing to reflexive collective intelligence: The IEML research program. *Information Sciences*, 180(1):71–94, 2010.

[23] Thomas W. Malone, Robert Laubacher, and Chrysanthos N. Dellarocas. Harnessing Crowds: Mapping the Genome of Collective Intelligence. *SSRN Electronic Journal*, 2009.

[24] Francis Heylighen. Collective Intelligence and its Implementation on the Web: Algorithms to Develop a Collective Mental Map. *Computational & Mathematical Organization Theory*, 5(3):253–280, 1999.

[25] Michael F Goodchild. Citizens as Voluntary Sensors: Spatial Data Infrastructure in the World of Web 2.0. *International Journal of Spatial Data Infrastructures Research*, 2:24–32, 2007.

[26] Kari A. Hintikka and Kari A. Web 2.0 and the collective intelligence. In *Proceedings of the 12th international conference on Entertainment and media in the ubiquitous era - MindTrek '08*, page 163. ACM, 2008.

[27] Dawn G. Gregg and Dawn G. Designing for collective intelligence. *Communications of the ACM*, 53(4):134, 2010.

[28] Dimitrios J. Vergados, Ioanna Lykourentzou, and Epaminondas Kapetanios. A resource allocation framework for collective intelligence system engineering. In *Proceedings of the International Conference on Management of Emergent Digital EcoSystems - MEDES '10*, page 182. ACM, 2010.

[29] Tim O'Reilly. What Is Web 2.0 - O'Reilly Media, 2005. URL `http://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html`. Last visited at 13.11.2018.

[30] Plerre-P. Grassé. La reconstruction du nid et les coordinations inter-individuelles chez Bellicositermes natalensis et Cubitermes sp. La théorie de la stigmergie: Essai d'interprétation du comportement des Termites constructeurs. *Insectes Sociaux*, 6 (1):41–80, 1959.

[31] Juergen Musil, Angelika Musil, Danny Weyns, and Stefan Biffl. An Architecture Framework for Collective Intelligence Systems. In *2015 12th Working IEEE/IFIP Conference on Software Architecture*, pages 21–30. IEEE, 2015.

[32] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Auton Agent Multi-Agent Syst*, 17:432–456, 2008.

[33] Angelika Musil, Juergen Musil, and Stefan Biffl. Major variants of the SIS architecture pattern for collective intelligence systems. In *Proceedings of the 21st European Conference on Pattern Languages of Programs - EuroPlop '16*, pages 1–11. ACM, 2016.

[34] Elisa Y. Nakagawa, Marcelo Gonçalves, Milena Guessi, Lucas B. R. Oliveira, and Flavio Oquendo. The state of the art and future perspectives in systems of systems software architectures. In *Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems - SESoS '13*, pages 13–20. ACM, 2013.

[35] Eduardo Silva, Everton Cavalcante, Thais Batista, Flavio Oquendo, Flavia C. Delicato, and Paulo F. Pires. On the Characterization of Missions of Systems-of-Systems. In *Proceedings of the 2014 European Conference on Software Architecture Workshops - ECSAW '14*, pages 1–8. ACM, 2007.

[36] Mark W. Maier. Architecting Principles for Systems-of-Systems. *INCOSE International Symposium*, 6(1):565–573, 1996.

[37] Charles Keating, Ralph Rogers, Resit Unal, David Dryer, Andres Sousa-Poza, Robert Safford, William Peterson, and Ghaith Rabadi. System of systems engineering. *EMJ - Engineering Management Journal*, 15(3):36–45, 2003.

[38] Gary D. Wells and Andrew P. Sage. Engineering of a System of Systems. In *System of Systems Engineering: Innovations for the 21st Century*, chapter 3, pages 44–76. John Wiley & Sons, Inc., 2009.

[39] Claire Ingram, Richard Payne, Simon Perry, Jon Holt, Finn Overgaard Hansen, and Luis Diogo Couto. Modelling patterns for systems of systems architectures. In *2014 IEEE International Systems Conference Proceedings*, pages 146–153. IEEE, 2014.

[40] Soumya Simanta, Edwin Morris, Grace A. Lewis, and Dennis B. Smith. Engineering lessons for systems of systems learned from service-oriented systems. In *IEEE International Systems Conference Proceedings*, pages 634–639. IEEE, 2010.

[41] Pal Varga, Fredrik Blomstedt, Luis Lino Ferreira, Jens Eliasson, Mats Johansson, Jerker Delsing, and Iker Martínez de Soria. Making system of systems interoperable – The core components of the arrowhead framework. *Journal of Network and Computer Applications*, 81:85–95, 2017.

[42] Cor-Paul Bezemer and Andy Zaidman. Multi-tenant SaaS applications: maintenance dream or nightmare? In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE) on - IWPSE-EVOL '10*, page 88. ACM, 2010.

[43] Jm Kaplan. Saas: Friend or foe? *Business Communications Review*, 37(June):48, 2007. URL `http://www.webtorials.net/main/resource/papers/BCR/paper125/06kaplan.pdf`. Last visited at 11.09.2018.

[44] Haitham Yaish, Madhu Goyal, and George Feuerlicht. An elastic multi-tenant database schema for software as a service. In *Proceedings - IEEE 9th International Conference on Dependable, Autonomic and Secure Computing, DASC 2011*, pages 737–743. IEEE, 2011.

[45] Sanjukta Pal, Amit Kr Mandal, and Anirban Sarkar. Application Multi-Tenancy for Software as a Service. *ACM SIGSOFT Software Engineering Notes*, 40(2):1–8, 2015.

[46] Sumit Kalra and T. V. Prabhakar. Patterns for Managing Tenants in a Multi-tenant Application. In *Proceedings of the 22nd European Conference on Pattern Languages of Programs - EuroPLoP '17*, pages 1–10. ACM, 2017.

[47] Bret Waters. Software as a service: A look at the customer benefits. *Journal of Digital Asset Management*, 1(1):32–39, 2005.

[48] Hailue Lin, Kai Sun, Shuan Zhao, and Yanbo Han. Feedback-Control-Based Performance Regulation for Multi-Tenant Applications. In *2009 15th International Conference on Parallel and Distributed Systems*, pages 134–141. IEEE, 2009.

[49] Chang Jie Guo, Wei Sun, Ying Huang, Zhi Hu Wang, and Bo Gao. A Framework for Native Multi-Tenancy Application Development and Management. In *The 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (CEC-EEE 2007)*, pages 551–558. IEEE, 2007.

[50] Frederick Chong, Gianpaolo Carraro, and Roger Wolter. Multi-Tenant Data Architecture. 2006. URL `http://ramblingsofraju.com/wp-content/uploads/2016/08/Multi-Tenant-Data-Architecture.pdf`. Last visited at 15.07.2018.

[51] Gozde Karatas, Ferit Can, Gamze Dogan, Cemile Konca, and Akhan Akbulut. Multi-tenant architectures in the cloud: A systematic mapping study. In *2017 International Artificial Intelligence and Data Processing Symposium (IDAP)*, pages 1–4. IEEE, 2017.

[52] Zhi Hu Wang, Chang Jie Guo, Bo Gao, Wei Sun, Zhen Zhang, and Wen Hao An. A Study and Performance Evaluation of the Multi-Tenant Data Tier Design Patterns for Service Oriented Computing. In *IEEE International Conference on e-Business Engineering*, pages 94–101. IEEE, 2008.

[53] R Krebs, Christof Momm, and Samuel Kounev. Architectural Concerns in Multi-tenant SaaS Applications. *Closer*, pages 426–431, 2012.

[54] Craig D. Weissman and Steve Bobrowski. The design of the force.com multitenant internet application development platform. In *Proceedings of the 35th SIGMOD international conference on Management of data - SIGMOD '09*, page 889. ACM, 2009.

[55] Xiaoyan Jiang, Yong Zhang, and Shijun Liu. A Well-designed SaaS Application Platform Based on Model-driven Approach. In *2010 Ninth International Conference on Grid and Cloud Computing*, pages 276–281. IEEE, 2010.

[56] Israel Herraiz, Daniel Rodriguez, Gregorio Robles, and Jesus M. Gonzalez-Barahona. The evolution of the laws of software evolution. *ACM Computing Surveys*, 46(2): 1–28, 2013.

[57] M M Lehman, J F Ramil, and G Kahen. Evolution as a Noun and Evolution as a Verb. *Workshop on Software and Organisation Co-evolution (SOCE)*, 2000.

[58] Priyadarshi Tripathy and Kshirasagar Naik. *Software Evolution and Maintenance: A Practitioner's Approach*. John Wiley & Sons, Inc., 2014.

[59] M.M. Lehman. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, 1980.

[60] M. M. Lehman. Laws of software evolution revisited. In *Proceedings of the 5th European Workshop on Software Process Technology*, volume 1149, pages 108–124. Springer, 1996.

[61] V.T. Rajlich and K.H. Bennett. A staged model for the software life cycle. *IEEE Computer*, 33(7):66–71, jul 2000.

[62] Václav Rajlich and Václav. Software evolution and maintenance. In *Proceedings of the on Future of Software Engineering - FOSE 2014*, pages 133–144. ACM, 2014.

[63] Keith H. Bennett and Václav T. Rajlich. Software maintenance and evolution. In *Proceedings of the conference on The future of Software engineering - ICSE '00*, pages 73–87. ACM, 2000.

[64] Isabelle Côté, Maritta Heisel, and Jeanine Souquières. On the Evolution of Component-Based Software. pages 54–69. Springer, 2012.

[65] N. Sadou, D. Tamzalit, and M. Oussalah. A unified approach for software architecture evolution at different abstraction levels. In *Eighth International Workshop on Principles of Software Evolution (IWPSE'05)*, pages 65–68. IEEE, 2005.

[66] Qianxiang Wang, Junrong Shen, Xiaopeng Wang, and Hong Mei. A component-based approach to online software evolution. *Journal of Software Maintenance and Evolution: Research and Practice*, 18(3):181–205, 2006.

[67] Jeffrey M. Barnes, David Garlan, and Bradley Schmerl. Evolution styles: foundations and models for software architecture evolution. *Software & Systems Modeling*, 13(2): 649–678, 2014.

[68] Mikio Aoyama. Metrics and analysis of software architecture evolution with discontinuity. In *Proceedings of the International Workshop on Principles of Software Evolution*, IWPSE '02, pages 103–107. ACM, 2002.

[69] Mikio Aoyama. Continuous and discontinuous software evolution: Aspects of software evolution across multiple product lines. In *Proceedings of the 4th International Workshop on Principles of Software Evolution*, IWPSE '01, pages 87–90. ACM, 2001.

[70] Seung-Pyo Jun, Hyoung Sun Yoo, and San Choi. Ten years of research change using Google Trends: From the perspective of big data utilizations and applications. *Technological Forecasting and Social Change*, 130:69–87, 2018.

[71] Hyunyoung Choi and Hal Varian. Predicting the Present with Google Trends. *Economic Record*, 88:2–9, 2012.

[72] N Ren, Z Fang, H Sun, B Sun, and Yang Zhao. CSCW based customer relationship management system. In *The 6th International Conference on Networked Computing and Advanced Information Management*, pages 530–535, 2010.

[73] James D. Palmer and N. Ann Fields. Computer-Supported Cooperative Work. *IEEE Computer*, 27(5):15–17, 1994.

[74] Uwe M Borghoff and Johann H Schlichter. *Computer-Supported Cooperative Work: Introduction to Distributed Applications*. Springer, 2000.

[75] Nick Ismail. CRM will be the fastest growing software market in 2018, 2018. URL `http://www.information-age.com/crm-fastest-growing-software-market-2018-gartner-123471378/`. Last visited at 29.05.2018.

[76] Michael Maoz and Brian Manusama. Magic Quadrant for the CRM Customer Engagement Center, 2017. URL `https://www.gartner.com/doc/reprints?id=1-3XZENPT{&}ct=170414{&}st=sb`. Last visited at 29.05.2018.

[77] Val Swisher. Glossary Versus Terminology - What's the Difference?, 2014. URL `http://contentrules.com/glossary-versus-terminology-whats-difference/`. Last visited at 05.08.2018.

[78] Laura Brandenburg. The Glossary: A Gateway to Clear Requirements and Communication. URL `http://www.bridging-the-gap.com/glossary/`. Last visited at 06.08.2018.

[79] Sanjay Kumar Dubey and Ajay Rana. A comprehensive assessment of object-oriented software systems using metrics approach. *International Journal on Computer Science and Engineering*, 2:2726–2730, 11 2010.

[80] Colin Robson. *Real World Research: A Resource for Users of Social Research Methods in Applied Settings*. John Wiley & Sons Ltd, third. edition, 2011.

[81] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.

[82] Claes Wohlin, Martin Höst, and Kennet Henningsson. *Empirical Research Methods in Software Engineering*. Springer, Berlin, Heidelberg, 2003.

[83] David V Thiel. *Survey research methods*, pages 192–229. Cambridge University Press, 2014.

[84] Timothy C. Lethbridge, Susan Elliott Sim, and Janice Singer. Studying Software Engineers: Data Collection Techniques for Software Field Studies. *Empirical Software Engineering*, 10(3):311–341, 2005.

[85] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Springer, 2012.