Lukas Tanner BSc

Design and Integration of an Instant Messaging System in an Enterprise Industry Automation Software.

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Dipl.-Ing. Dr.techn. Georg Macher BSc
Dipl.-Ing. Michael Krisper BSc

Institute of Technical Informatics

Dipl.-Ing. Christoph Scherr BSc
Siemens AG Austria

Graz, January 2020

AFFIDAVIT

Abstract

Increasingly complex processes in industrial automation result in more requirements on the collaboration between teams and companies. The need for a unified communication process arises, as nowadays, automation engineers working on programmable logic controllers (PLCs) can be distributed on several buildings, countries or collaborate with engineers from various subsidiary companies. One way of providing a unified communication process is by using a dedicated instant messaging system (IMS). While several studies elaborate on the controversial usage of IMSs within work, the majority agrees on the positive effects.

The automation customers of Siemens use their enterprise industry automation software, *Totally Integrated Automation Portal* (TIA Portal). Currently, the TIA Portal does not provide a communication mechanism for collaborating automation engineers. In practice, we have seen that this causes delays and misconceptions due to the usage of various communication tools. Hence, this thesis focuses on how to design and integrate an IMS into the TIA Portal to solve those issues.

Within the scope of this thesis, three different approaches on how an IMS can be integrated have been implemented and evaluated. Approach 1 enables instant messaging through WebRTC. WebRTC allows peer-to-peer communication within a browser. The signaling server required for connection establishment is implemented in Node.js. Approach 2 is a cloud-based solution building on Microsoft's SignalR service. SignalR is used to add real-time web functionality to the underlying ASP.NET Core application. Approach 3 enables instant messaging via connected PLCs. For this purpose, the on-board functionality of Siemens Simatic PLCs is used. Evaluation of the different approaches shows that approach 1 is the primary choice with regards to the requirements defined by Siemens.

Keywords: Instant Messaging, WebRTC, Signaling, Node.js, SignalR, ASP.NET Core, PLC, Azure

Kurzfassung

Immer komplexere Prozesse in der industriellen Automatisierung führen zu höheren Anforderungen an die Zusammenarbeit zwischen Teams und Unternehmen. Der Bedarf an einem einheitlichen Kommunikationsprozess entsteht, da Automatisierungsingenieure, die an speicherprogrammierbaren Steuerungen arbeiten, heutzutage verteilt in verschiedenen Gebäuden und Ländern oder mit Ingenieuren verschiedener Tochtergesellschaften zusammenarbeiten. Eine Möglichkeit, einen einheitlichen Kommunikationsprozess bereitzustellen, besteht in der Verwendung eines dedizierten Instant-Messaging-Systems (IMS). Zahlreiche Studien befassten sich bereits mit den Auswirkungen eines IMS am Arbeitsplatz. Obwohl einige Studien die negativen Auswirkungen darstellen, ist sich die Mehrheit der Studien über die positiven Auswirkungen einig.

Automatisierungskunden von Siemens nutzen das Engineering-Framework *Totally Integrated Automation Portal* (TIA Portal) für die digitalisierte Automatisierung. Derzeit bietet das TIA Portal keinen Kommunikationsmechanismus für Automatisierungsingenieure, die zusammen an einer Automatisierungslösung arbeiten. In der Praxis hat sich gezeigt, dass verschiedene Kommunikationsmittel verwendet werden und dies zu Verzögerungen oder gar Missverständnissen führt. Daher befasst sich diese Masterarbeit mit dem Design und der Integration eines IMS in das TIA Portal um die zuvor erwähnten Probleme zu lösen.

Im Rahmen dieser Arbeit wurden drei verschiedene Ansätze zur Integration eines IMS implementiert und evaluiert: Ansatz 1 ermöglicht Instant Messaging über WebRTC. WebRTC stellt Echtzeitkommunikation innerhalb eines Browsers für sogenannte Peers bereit. Der für den Verbindungsaufbau erforderliche Signalisierungsserver ist in Node.js implementiert. Ansatz 2 ist eine Cloud-basierte Lösung, die auf dem SignalR Dienst von Microsoft aufbaut. SignalR wird verwendet, um der zugrunde liegenden ASP.NET Core-Anwendung Echtzeitkommunikation zu ermöglichen. Ansatz 3 ermöglicht Instant Messaging über eine On-Board-Funktionalität verbundener Siemens Simatic speicherprogrammierbarer Steuerungen. Die Evaluierung der verschiedenen Ansätze zeigt, dass Ansatz 1 die bevorzugte Lösung im Hinblick auf die von Siemens definierten Anforderungen ist.

Acknowledgements

I want to thank my supervisors Georg Macher and Michael Krisper, for the valuable and fast feedback they provided. Their ideas and input helped a lot in implementing the different approaches used in this thesis and increasing the quality with regards to the content.

I also want to thank several people at Siemens. I owe my thanks, Christoph Scherr, for giving me the chance to write my thesis at Siemens, where I was able to work on a practice-oriented problem. Thanks to my former team Istari for their support throughout the year and their jokes that I will never finish my thesis - yes, I finally did it. Moreover, thanks to Christian Eitner, for teaching me the essentials of writing a thesis with his incredible patience.

I am deeply thankful for my family. They always lent me a sympathetic ear and encouraged me to finish my studies. I could not imagine a better family. Finally, I also want to thank the love of my life, Michi, for always backing me up in hard times. Studying is not only fun, it also involves stress and captures a lot of free time. You are the strong woman behind a man that everybody aims for. I am really looking forward to start a new chapter in our lives.

Lukas Tanner

Contents

1 Introduction **11**

2 Background **13**
2.1 Fundamentals 13
2.1.1 Cloud computing 13
2.1.2 Web Service 15
2.1.3 Virtualization and Containerization 17
2.1.4 Edge Computing 20
2.2 Instant Messaging 21
2.3 Instant Messaging Architectures 23
2.3.1 Client-Server 23
2.3.2 Peer-to-Peer 24
2.3.3 Serverless 29
2.4 Security Aspects of Instant Messaging 31
2.4.1 Security Requirements of Instant Messaging 31
2.4.2 Security Threats of Instant Messaging 32
2.5 Basic Instant Messaging Protocols 35
2.5.1 XMPP 35
2.5.2 SIP/SIMPLE 38
2.6 Real-Time Communication 41
2.6.1 WebRTC 42
2.6.2 SignalR 43
2.7 Graphical User Interface Design 44

3 Related Work on Instant Messaging **46**
3.1 Positive Work-Related IM Effects 46
3.2 Negative and Controversial Work-Related IM Effects 47

4 Problem Statement **49**
4.1 Problem Description 49
4.2 Requirements 50

5 Related Industrial Standards **52**
5.1 Standard ISA 95 52
5.2 Standard ISA 88 53
5.3 Standard IEC 61131 53
5.4 Standard IEC 62264 53
5.5 Standard IEC 61508 54

6 Industrial Environment **55**

7 Approach **57**

7.1 Approach 1: WebRTC with Node.js Server 57
7.2 Approach 2: Cloud-based solution with SignalR service 60
7.3 Approach 3: Chat via PLC 64

8 Evaluation 68
8.1 Discussion of Approach 1 71
8.2 Discussion of Approach 2 71
8.3 Discussion of Approach 3 71
8.4 Result 72

9 Limitations 73
9.1 Compatibility/Technology/Dependencies 73
9.2 Scalability 73
9.3 Usability 74
9.4 Functionality 74
9.5 Process/Maintenance 74

10 Conclusion and Future Work 75

Acronyms 76

References 78

List of Figures

1 Cloud service models 15
2 Virtual machine system structure 17
3 Virtual machine monitor architectures 18
4 Virtualization vs. containerization 19
5 Edge/Fog computing overview 20
6 Instant messaging services 21
7 Presence system 22
8 Instant messaging system 22
9 Client-server Architecture 23
10 Replicated-server Architecture 24
11 Pure Peer-to-peer Architecture 26
12 Hybrid Peer-to-peer Architecture 27
13 Super Peer-to-Peer Architecture 27
14 Exemplary DHT lookup query 28
15 Serverless services overview 29
16 Overview of most important FaaS providers 30
17 XMPP architecture 35
18 Simplified SIP session establishment 39
19 SIMPLE presence information exchange 39
20 SIMPLE message exchange 40
21 STUN and TURN server concept 43
22 Design requirements of different user experience levels 45
23 Automation pyramid 53
24 Exemplary automation setup 56
25 Approach 1 overview 58
26 Approach 1 signaling 59
27 Approach 1 GUI 60
28 Approach 2 overview 61
29 Approach 2 message transfer 62
30 Approach 2 GUI 64
31 Approach 3 overview 65
32 Approach 3 workflow 66
33 Approach 3 GUI 67

Listings

1 WSDL 2.0 example definition 16
2 XMPP XML stream example 36
3 Exemplary JavaScript code used to connect to easyRTC server . 58
4 Exemplary SignalR Hub code 63
5 SignalR connection establishment 63

List of Tables

1 Functional requirements 50
2 Non-functional requirements 51
3 Feature Matrix 70

1 Introduction

Increasingly complex processes in industrial automation result in more requirements on the collaboration between teams and companies. One of such requirements is to learn how to effectively communicate through various time zones, locations, and cultures (Reed & Knight, 2010). The mix of different communication technologies like phone calls, emails, forums, instant messaging (IM), etc., can lead to communication bottlenecks and thus decreases the productivity. However, it has been shown (Ou, Davison, Liang, & Zhong, 2010; Bakar & Johari, 2009) that IM is an effective communication tool and is particularly helpful in communication of geographically separated teams (Armoogum & Mudhoo, 2016).

The automation customers of Siemens use their enterprise industry automation software, *Totally Integrated Automation Portal* (TIA Portal). Currently, the TIA Portal does not provide a communication mechanism for collaborating automation engineers. More information about the problems that occur without a unified communication process can be found in Chapter 4. The goal of this thesis is to design and implement an IMS that is integrated into the TIA Portal and thus, provides a unified communication process.

Implementing an IMS involves several decisions, such as the underlying architecture, security requirements, protocols, or graphical user interface (GUI). Traditional IM architectures are client-server or peer-to-peer (Barry & Tom, 2011) and use the standardized communication protocols session initiation protocol (SIP) (Campbell, Rosenberg, Schulzrinne, Huitema, & Gurle, 2002) or the extensible messaging and presence protocol (XMPP) (Saint-Andre, 2009). With the rise of cloud computing, new possibilities like serverless architectures have emerged. Serverless architecture refers to a programming model where small code snippets are executed in the cloud without any control over the resources on which the code runs (Baldini et al., 2017). Serverless architectures gained increased popularity due to the recent shift of enterprise applications to containers and micro-services (Baldini et al., 2017).

Adding real-time-communication (RTC) functionality to an application used for IM is an essential task. Too long message delays can decrease the acceptance of an IMS or even result in abandonment. Two important concepts exist for enabling RTC that are used within the scope of this thesis: web real-time communication (WebRTC) and SignalR. WebRTC is a collection of standards and protocols for enabling audio/video calling, video chats, and P2P chat for browser-to-browser applications without needing any third-party software. SignalR is a framework from Microsoft targeting ASP.NET used to integrate RTC in web applications.

As IM has increasingly become the target of attacks (Armoogum & Mudhoo, 2016), adequate security mechanisms must be ensured (Barry & Tom, 2011). Unger et al. (2015) identified the following three key challenges that need to be considered: trust establishment, conversation security, and transport privacy. Additionally, Abu-Salma, Sasse, Bonneau, and Smith (2015) argue that for achieving adequate security, messaging tools need a high level of usability. However, Musiani and Ermoshina (2017) observed a trade-off between security and usability in most of the existing IM tools.

The remainder of this work is structured as follows: Chapter 2 provides background information on IM needed within the scope of this thesis. Work-related IM effects are discussed in Chapter 3. Chapter 4 explains the underlying problem statement and requirements. Chapter 5 outlines the industrial setting with focus on important standards. Chapter 6 illustrates the environment of the target enterprise industry automation software, the TIA Portal. In Chapter 7, the implemented approaches are presented. An evaluation of implemented approaches is done in Chapter 8. Limitations of the approaches are discussed in Chapter 9. Finally, Chapter 10 gives a conclusion and an outlook on possible future work.

2 Background

This chapter provides an introduction to IMSs. The first section covers the fundamental technologies used in this thesis. Afterwards, a definition of IMS is provided. In the following sections, basic architectures, protocols, and IMS security aspects are explained. Finally, the last sections focus on examining real-time communication and graphical user interface (GUI) design.

2.1 Fundamentals

This section aims to explain basic technologies and mechanisms that are used throughout this thesis. It covers cloud computing, the definition of a web service, virtualization, containerization, and edge/fog computing.

2.1.1 Cloud computing

Over the last years, cloud computing became an vital technology trend (Furht, 2010). A definition for cloud computing is provided by the NIST institute (Mell & Grance, 2011): *"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."* The base concept of cloud computing is the cloud. In general, the term cloud is used the describe the conglomeration of servers, where processing power, storage, and information is made available for users over the internet. Clouds can be classified into four different types:

a) *Public cloud*: This is the most common type of cloud computing. Customers can access publicly available services over the internet from a third-party service provider.
b) *Private cloud*: A private cloud refers to a cloud exclusively used by a single organization or individual. On-premise is a variant where the organization itself is responsible for the execution and management of the services.
c) *Community cloud*: In a community cloud, the cloud infrastructure is managed by a community of consumers.
d) *Hybrid cloud*: Is a combination of two or more distinct cloud types.

Benefits

In comparison to traditional computing concepts, cloud computing offers a vast variety of new features (Wang et al., 2008; Ramadan & Kashyap, 2010):

a) *Scalability*: Resources and services are provided on demand. The cloud provider can easily scale assigned resources depending on the workload.
b) *Guaranteed Quality of Service (QoS)*: The cloud provider guarantees a high service level towards performance and availability.
c) *Mobility*: Services can be used location independent.
d) *Costs*: Users do not need to afford and set up expensive infrastructure, because the cloud provider already does it. Further, they only need to pay for the needed services and storage capacity.
e) *Autonomous system*: Cloud computing systems are transparently managed autonomous systems. In order to present a simple platform depending on the user's requirements, data within the cloud can be reconfigured or consolidated automatically.

Drawbacks

In contrast to the benefits, there are also several drawbacks. In essence, the biggest concern is the loss of control and the security and privacy issues that go hand in hand with it (Ramadan & Kashyap, 2010). Using a cloud provider implies a complete data handover introducing the risk of unauthorized data access. Jadeja and Modi (2012) argue that also replication time and reliability can be an issue. A high replication time can harm data resiliency. Cloud server downtimes contribute to reliability.

Service models

Another way to describe cloud computing is to have a look at the offered service models. Cloud computing is composed of three service models (Mell & Grance, 2011) that are illustrated in figure 1. The on-premise approach refers to a concept where all of the services, resources, and applications are self-managed. The three service models are defined as:

a) *Infrastructure as a service (IaaS)*: IaaS means that computer infrastructure, e.g., server, computer, disk space, network, is provided as an outsourced service (Chang, Abu-Amara, & Sanford, 2010). Resources can be acquired dynamically when they are needed.

b) *Platform as a service (PaaS)*: PaaS builds on top of IaaS. Services provided by IaaS are extended with a custom software stack for specific applications, e.g., database or operating system (Furht, 2010).

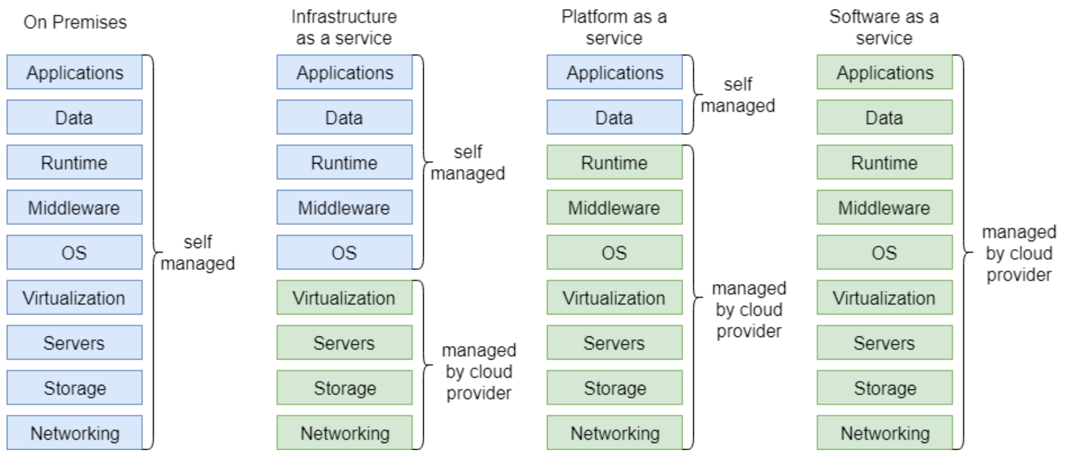


Figure 1: Cloud service models adapted from Microsoft Docs (2018)

c) *Software as a service (SaaS)*: SaaS means that whole applications are running in a cloud infrastructure and are provided to customers (Buxmann, Hess, & Lehmann, 2008). The SaaS provider is responsible for running and maintaining the offered applications. Clients use different devices to access applications. SaaS runs on top of PaaS.

Another service model referring to a new PaaS generation is function as a service (FaaS). FaaS is used in the serverless paradigm. A detailed explanation of serverless and FaaS is provided in Section 2.3.3. One important aspect in terms of cloud computing are enabling technologies, building the cloud computing fundament. Besides the basic Web 2.0 technology, other enabling technologies are web services and virtualization (Wang et al., 2010). In the following sections, web services, virtualization, and containerization are defined.

2.1.2 Web Service

IBM defines a web service as *"self-contained, self-describing, modular applications that can be published, located, and invoked across the web. Web services perform functions that can be anything from simple requests to complicated business processes"* (Tidwell, 2000). Web services can be categorized into *stateless* and *stateful* services (Papazoglou & Dubray, 2004). When a service is invoked repeatedly without the need to maintain any context or state, the service is considered to be stateless, while services that need to preserve context information are considered to be stateful. Cloud services are typically designed as web services, which follow industry standards such as SOAP or REST (Furht, 2010). Web services description language (WSDL) is an XML-based interface description language used to describe web services. It addresses the problem of providing a uniform and consistent web service description. The latest version of the standard is WSDL 2.0 and was published in 2007 (Chinnici, Moreau, Ryman, & Weerawarana, 2007). WSDL 2.0 enables the description separation of functionality from the concrete details, such as the location of the offered functionality. WSDL files are used for the service description (Josuttis, 2007). They consist of used data types descriptions and three additional layers: i) Interface layer: Describes the provided operations with input and output parameters that use the defined data types. ii) Binding layer: Contains the web service's protocol and format. iii) Service layer: Uses the address or URL of the web service the specify its location.

An exemplary WSDL 2.0 web service definition is shown in listing 1. It contains a single interface description with one input and output parameter and uses two defined types.

```
<description>
  <types>
      ...
      <xs:element name="latestRequest" type="requestType"/>
      <xs:complexType name="requestType">
        <xs:sequence>
          <xs:element name="date" type="xs:date"/>
        </xs:sequence>
      </xs:complexType>
      <xs:element name="responseType" type="xs:string"/>
    </xs:schema>
  </types>
  <interface name = "myInterface" >
    <operation name="myOperation">
      <input messageLabel="In" element="stns:requestType" />
      <output messageLabel="Out" element="stns:responseType" />
    </operation>
  </interface>
  <binding>
  ...
  </binding>
  <service>
  ...
  </service>
</description>
```

Listing 1: WSDL 2.0 example definition

2.1.3 Virtualization and Containerization

Virtualization is a concept developed over the last decades and is especially important for cloud infrastructure. Pearce, Zeadally, and Hunt (2013) describe the essence of system virtualization as the use of a software layer encapsulating or underlying an operating system (OS) that provides the same expected behavior as the physical hardware. This layer is referred to as the virtual machine monitor (VMM) or hypervisor. The main difference between a VMM and an emulator is that all instructions are intercepted by the emulator, whereas a VMM only intercepts sensitive instructions. All non-sensitive instructions are executed directly on the hardware, if possible. Figure 2 shows the traditional structure of a virtual machine (VM) system. The VMM takes complete control of the machine hardware and creates virtual machines. The VMs behave like a complete physical machine running its own OS.

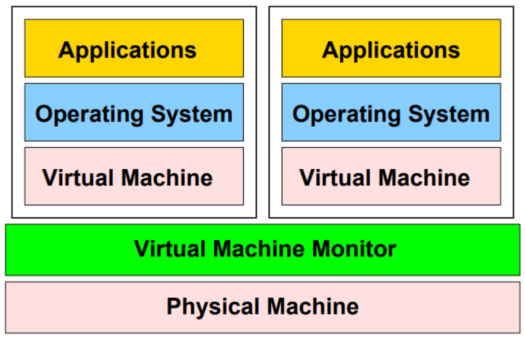


Figure 2: Virtual machine system structure by Sugerman, Venkitachalam, and Lim (2001).

There are two types of VMMs illustrated in figure 3. A *Type-I* (or classical) VMM is installed on the hardware as the primary boot system. Execution is done with the highest privilege level and full control over all used virtual machines. A *Type-II* (or hosted) VMM has a more complex structure. The VMM is located above or alongside a host OS and above the hardware. Drivers from the host operating system may be shared to handle input/output (I/O). Hardware-specific drivers are not required for VMM I/O operations, and using a VM with an existing environment is possible. This VMM type is used by popular products such as VMWare or Sandbox.

Virtualization offers various benefits (Menascé, 2005):

a) *Security*: Environments with different security requirements can be run on different virtual machines. Different tools and the OS can be chosen

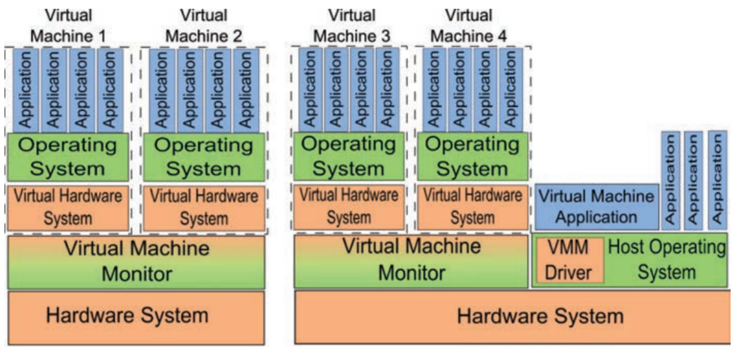


Figure 3: Virtual machine monitor architectures by Pearce, Zeadally, and Hunt (2013) : Type-I on the left and Type-II on the right.

to fulfill the security requirements.

b) *Reliability and Availability*: Software failures affect only the virtual machine the software is running on.

c) *Cost*: Several cost factors like hardware costs, space costs, software licenses can be reduced by server consolidation.

d) *Adaptability of workload variations*: Changing workload intensity levels are managed by resource shifting and priority allocations among virtual machines.

e) *Legacy applications*: Legacy applications can still be run on VMs. Hence, migration to another OS can be avoided.

Containerization

While virtualization provides many advantages, some limitations remain. The general use case of a VM is an isolated large file stored on the host's file system, run by a single, large process (Pahl, 2015). The need for full OS images for each VM together with the specific necessary binaries for the application contributes to disk storage shortness and a slow start-up. Out of the need for producing a more lightweight computation resource, containerization emerged. A container is defined as a package containing all necessary business logic, middleware, or ready-to-deploy application parts that are needed to run an application (Pahl, 2015). Containers work in a Linux environment providing isolation and efficient resource management. The basic principle is that containers share an OS and optionally binaries and libraries, resulting in smaller-sized deployments compared to hypervisor deployments (Bernstein, 2014). This enables the storage of numerous containers on a single physical host. Figure 4 illustrates the basic concept of containers compared to hypervisor-based virtualization. As containerization works at the OS level, it provides weaker isolation compared to virtualization. It is also limited in the number of supported guest OSs, e.g., Windows cannot be booted on Linux containers (Dua, Raja, & Kakadia, 2014). From the user's point of view, each container appears like a stand-alone OS.

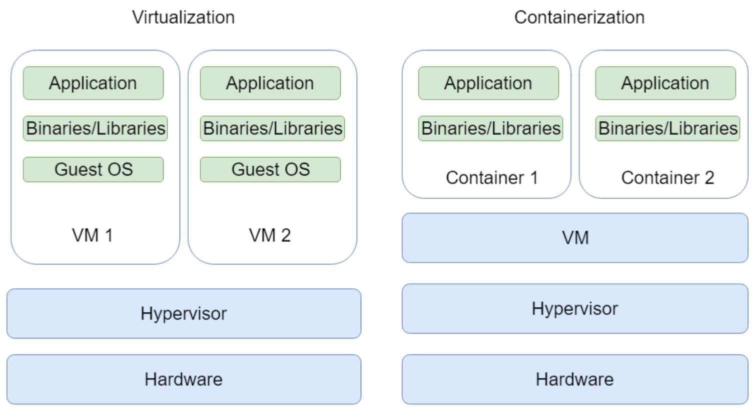


Figure 4: Virtualization vs. containerization adapted from Dua, Raja, and Kakadia (2014)

Docker is currently the most popular container solution (Pahl, 2015). According to Anderson (2015), Docker addresses two problem areas for developers. Firstly, continuous integration and development: Due to the lightweight nature of Docker containers, various production environments can easily be build and tested on a developer machine. Docker containers provide a high portability level, helping to solve the classic "it only works on my machine" problem. They also lead to time reduction because the container setup is much faster than the traditional VM setup. Secondly, capacity: VMs use a hypervisor, which accounts for 10-15 percent of the host's capacity. Docker containers need less memory and are up to 26 times faster than a VM.

Docker containers are constructed using base images. Docker uses an *onion* principle, meaning that each action taken within the container, e.g., installing a library, adds a new layer to the base image. Actions can be performed manually or using Dockerfiles (Combe, Martin, & Di Pietro, 2016). A Dockerfile is a script containing instructions listed successively. Execution of instructions forms a new image. An image can include only OS fundamentals or application stack parts up to the full application stack. Further, images can be saved and reused in other containers. Portable application containers are realized using complete docker images, resulting in a lightweight and easily distributable deployment.

2.1.4 Edge Computing

While cloud computing provides many benefits (see Section 2.1.1), problems for latency and data-intensive applications can arise (Bilal, Khalid, Erbad, & Khan, 2018). The distance between edge devices and the cloud data center is usually very high. With the rising number of edge devices, high latency is one of the biggest challenges that need to be faced by end-to-end applications. An emerging viable solution is edge computing. Edge computing is defined as a type of distributed computing, allowing the computation to be performed at the edge of the network (Shi, Cao, Zhang, Li, & Xu, 2016). Edge refers to any computing and network resource between the edge devices and the cloud data center. The main difference to cloud computing is the location of data processing. With the presence of edge devices, the computation load of data centers is reduced because some request computation can be done locally without the intervention of the cloud (Dolui & Datta, 2017). As a result, latency is reduced, and real-time handling of a subset of requests is supported. Fog computing also refers to extending the computation to the edge of the network. The terms fog and edge computing are often used interchangeably, although they are not the same. The term fog computing is created by Cisco and was introduced in 2012 (Bonomi, Milito, Zhu, & Addepalli, 2012). In general, fog computing is a paradigm describing how the edge computation is carried out. It involves the addition of networking layers between the edge devices and the cloud. In figure 5, an overview of fog computing is illustrated. The edge layer comprises all edge devices, and computation is either directly processed on the device or transferred to a fog node. The fog nodes contained in the fog layer can preprocess data and transmit it to the cloud on demand (Yi, Li, & Li, 2015).

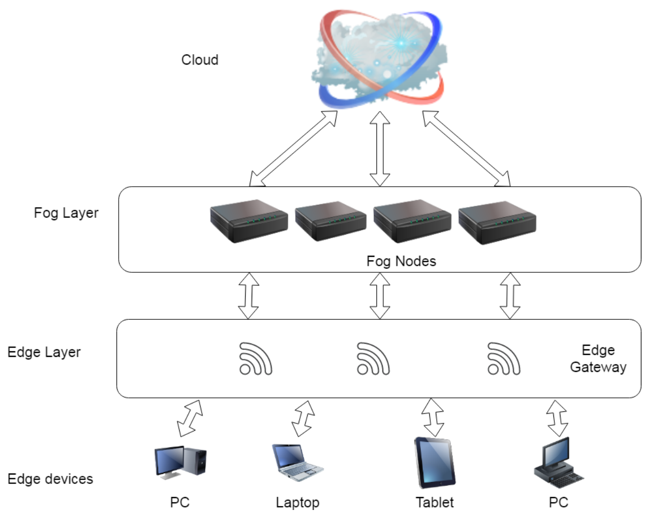
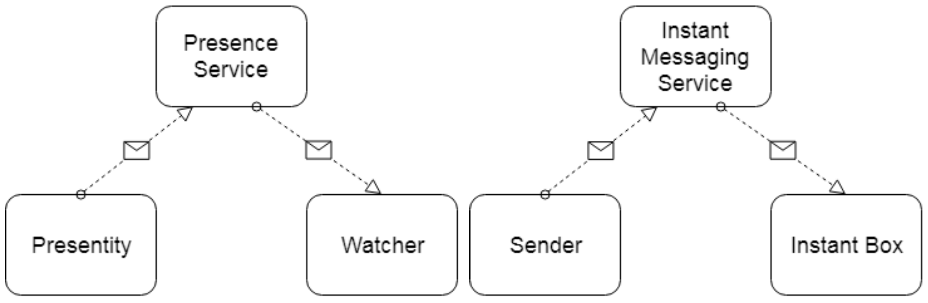


Figure 5: Edge/Fog computing overview

2.2 Instant Messaging

This section gives a definition of IM and the concept of an instant messenger. Instant messaging (IM) is defined as a type of computer-mediated communication (CMC). CMC subsumes communicative transactions using two or more networked computers (Barry & Tom, 2011). The main difference between other CMCs like email or social network services is that IM is intended for immediate end-user delivery (Debbabi & Rahman, 2003). According to the RFC2778 standard (Sugano, Day, & Rosenberg, 2000), IM comprises two basic services: *presence* and *instant messaging.* Figure 6 depicts an overview of the two services. The presence service is responsible for storing and distributing presence information. Presentities provide presence information. Watchers receive presence information via the service. The instant messaging service is responsible for delivering instant, real-time text messages from a sender to a particular instant inbox address.



(a) Presence service (b) Instant messaging service

Figure 6: Instant messaging services

In order to describe the general concept of an instant messenger, the terms presence system and IMS need to be explained before. In the following definitions, the term *principal* refers to real-world people or software using the system, and a user agent is a coupling between a principal and some system entity. A presence system comprises the presence service, a presence protocol, a presence user agent and watcher user agent for a single principal using a single presentity and a single watcher. Figure 7 shows a presence system. A typical example of a presence system is a *buddy-list* application, where the presence of users is exposed.

An IMS comprises the IM service, an IM protocol, a sender user agent, and inbox user agent for a single principal using a single sender and an instant inbox. Figure 8 shows the components of an IMS.

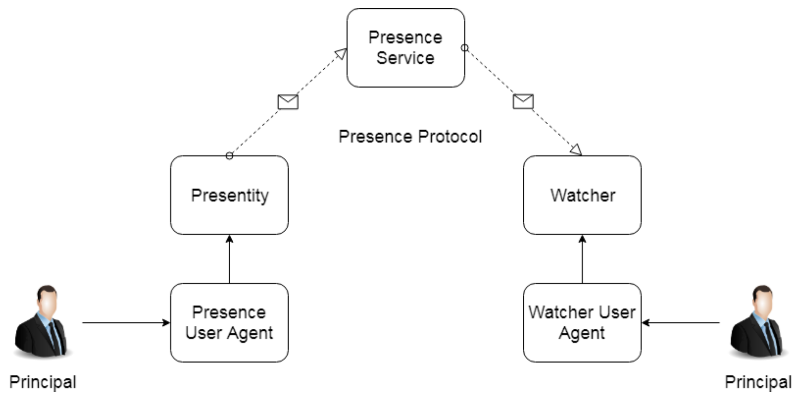


Figure 7: Presence system

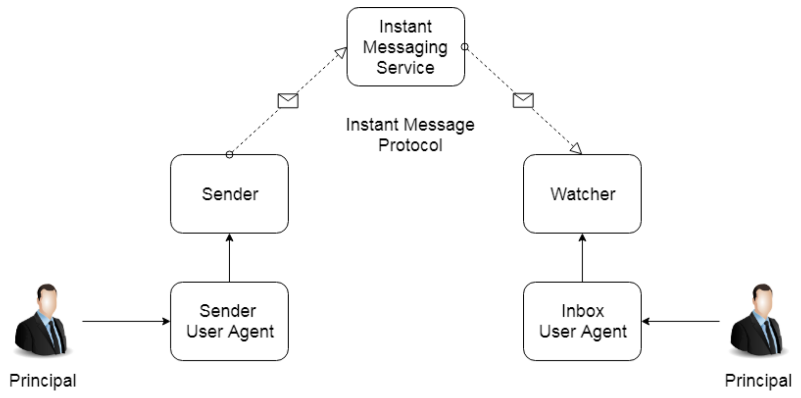


Figure 8: Instant messaging system

An instant messenger can then be defined as a combination of a buddy-list together with instant messaging capabilities provided by the IMS. Key features provided by an instant messenger are (Debbabi & Rahman, 2003; Mannan & van Oorschot, 2004; Garrett & Danziger, 2007):

- Sending and receiving of messages in real-time is supported.
- Either party can initiate communication.
- Different message formats (text, file, video) might be supported.
- Availability of users can be tracked via presence service.
- Message receivers are notified of incoming communication.

2.3 Instant Messaging Architectures

This chapter aims to give an overview of different architectures that are used in IMSs. Traditional architectures are client-server and peer-to-peer. Another architecture that can be used is serverless.

2.3.1 Client-Server

Client-server is a fundamental communication-model used in various application domains (Bischofs, Hasselbring, & Warns, 2008). Porter and Gough (2007) describe the roles in a client-server architecture as the following:

- Client: Requests specific services or resources.
- Server: Fulfills the requests.

Figure 9 shows a simple client-server system. Clients only interact with the server and not with any other peer. If clients want to communicate with each other, all information is routed over the server. This is often referred to as a centralized communication form (Williams & Ly, 2004).

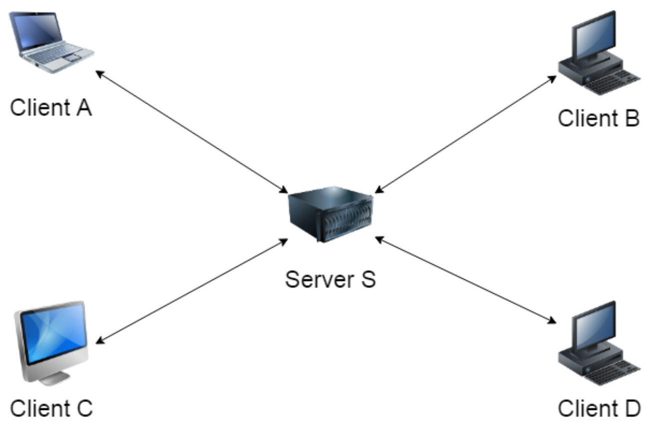


Figure 9: Client-server Architecture

Client-server architectures are categorized into symmetric and asymmetric architectures (Jennings et al., 2006). In a symmetric architecture, servers are responsible for the same functions. Hence, clients can use the servers interchangeably. This approach offers easy management but might not scale with large amounts of users. In order to support more massive networks, servers can also be replicated and interconnected (Williams & Ly, 2004). Figure 10 shows two separated networks, where clients registered at the head office server can communicate with clients registered at the branch office server.

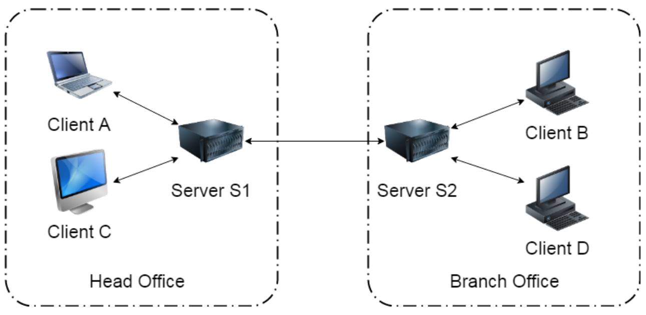


Figure 10: Replicated-server Architecture

On the contrary, in asymmetric architectures, a server is only responsible for a specific service, e.g., forwarding messages, or user discovery. This approach provides better a scalability for growing numbers of users but requires a complex infrastructure. According to Resig and Teredesai (2004), the formerly most popular IMSs (ICQ, MSN Messenger, Yahoo! Instant Messenger) and various IRC networks implemented a client-server architecture. A specific service provider administrated the server. Clients had to register themselves at the service provider and then use, e.g., the provider-approved client.

2.3.2 Peer-to-Peer

Peer-to-peer (P2P) is a fundamental communication-model used in various application domains (Bischofs et al., 2008). It means that peers communicate directly and can act as a server and client at the same time. Schollmeier (2001) defines the primary difference to the client-server model as the concept of a *servent*. Servents are entities that can act as a client or a server. Hauswirth and Dustdar (2005) characterize P2P systems via the following properties:

- Peers can act as a client or a server (servant).
- Decentralized and self-organized system.
- No central database exists.
- Peers act autonomously.
- Peers do not have a global view of the system - they only know the peers they are interacting with.

Formally, P2P systems are modelled as graphs (Bischofs et al., 2008). Nodes represent peers, and connections are modelled via edges between nodes. Connected peers are defined as neighbors. Peers can only exchange information with their neighbors. The RFC standard 5694 (Camarillo, 2009) describes two essential functions that a P2P system needs to provide:

a) *Enrollment Function:* The enrollment function is responsible for authenticating and authorizing new nodes that want to join the P2P system and for providing valid credentials.
b) *Peer Discovery Function:* The peer discovery function is responsible for allowing nodes to discover peers and connect to them. Joining a P2P system requires connection establishment to one or more peers.

Bootstrapping

The process of integrating a new node into a P2P system is referred to as bootstrapping (Cramer, Kutzner, & Fuhrmann, 2004). Therefore, bootstrapping aims to find a peer that is already connected to the network. Multiple discovery approaches have been defined (Dinger & Waldhorst, 2009):

1) *Bootstrap Server (or Static Overlay Node):* The basic idea of a bootstrap server is to provide addresses of already connected peers. These servers are endowed with well-known DNS names or IP addresses. Client software is shipped already containing the information about the active bootstrap servers. This approach was initially used by the file-sharing network Gnutella (Gnutella Developer Forum, 2003). In Gnutella, the bootstrap servers were called "pong caches" and provided information about the network topology and addresses of other peers. Pong caches returned a list of recent participating nodes on node joining requests. The first entry was used for establishing the connection.

2) *Local Host Cache:* Peers maintain a list with addresses of other peers. Rejoining the network works by connecting to peers from the cache without the need to contact a bootstrap server. This approach works as long as at least one node contained in the cache is active.

3) *Random Address Probing:* Nodes that want to join the network use randomly selected IP addresses and default ports to send the join request to active peers. This approach is not suited for a cold start of the P2P system.

4) *Network Layer Mechanisms:* Nodes can use, e.g., the network layer mechanism multicast for facilitating the bootstrap process. The problem with multicast is that multicast traffic is usually not routed beyond the local network domain.

P2P architectures

P2P networks build a virtual network, called overlay network, with its own routing mechanisms at the application level on top of the physical network (Kamel, Scoglio, & Easton, 2007). The overlay network contains all or only a subset of the physical nodes. Communication is carried out via TCP/IP, while peers can communicate directly at the application layer using the logical overlay links. Classification of P2P networks is done as unstructured or structured based on the link structure within the overlay network (Eberspächer & Schollmeier, 2005). Unstructured P2P architectures are categorized into pure, hybrid and super peer architectures (Jennings et al., 2006). Figure 11 shows a pure P2P architecture. Due to the absence of a server, this model is considered to be pure. This is often referred to as a decentralized communication form (Williams & Ly, 2004).

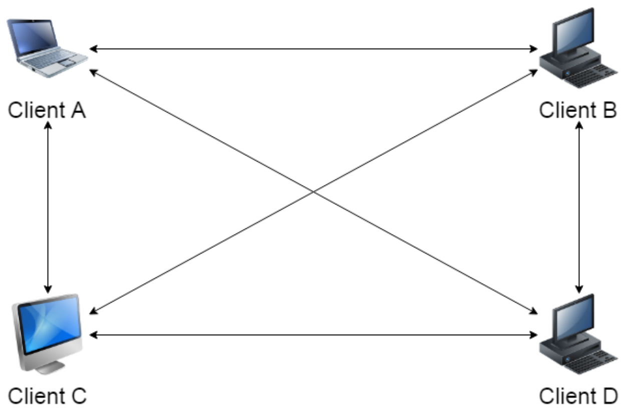


Figure 11: Pure Peer-to-peer Architecture

In hybrid P2P architectures, a central server, which is responsible only for a limited functional scope, is used. The central server is responsible for exchanging control and presence information. Peer communication is carried out like in the pure P2P architecture. Figure 12 shows an example of a hybrid P2P architecture.

Super peer architectures represent a combination of centralized and decentralized communication models (Bischofs et al., 2008). A super-peer is defined as a peer that serves as a server for a group of other peers while also serving as an equal in a network of super-peers (Yang & Garcia-Molina, 2004). A cluster contains a super-peer with all its assigned peers. The benefits of this approach are reduced time and bandwidth for search, load balancing, and server independence. Figure 13 shows an example of a super P2P architecture.

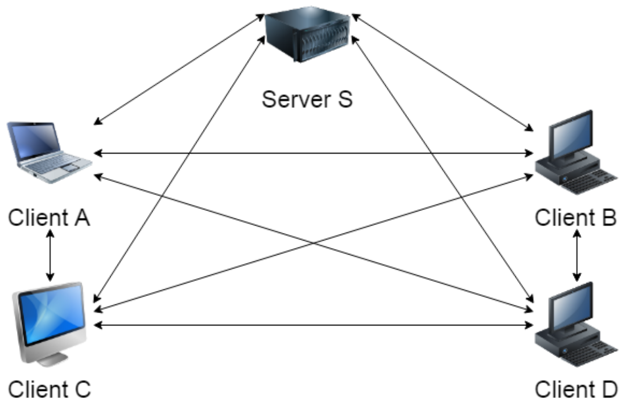


Figure 12: Hybrid Peer-to-peer Architecture

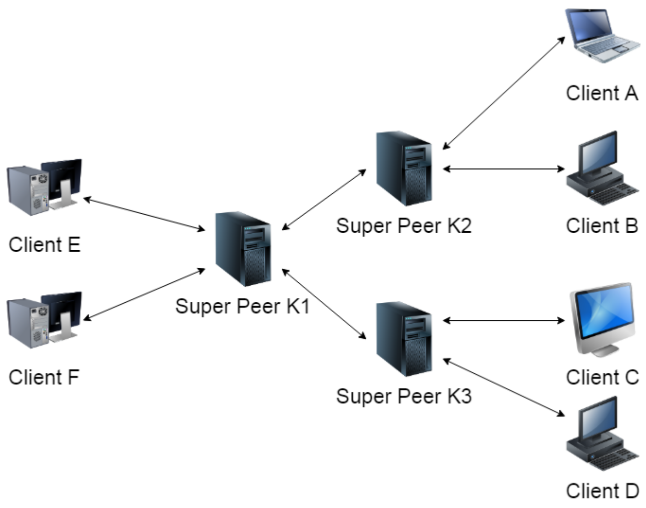


Figure 13: Super Peer-to-Peer Architecture

Distributed Storage and Lookup

In structured P2P architectures, the overlay network is tightly controlled, and resources are placed at specified locations (Androutsellis-Theotokis & Spinellis, 2004). A mapping between the content and location is provided via a distributed hash table (DHT) in order to provide efficient routing. A DHT consists of a lookup and storage protocol (Sit & Morris, 2002). The essential components of the lookup protocol are the key and node identifier space, rules for associating keys to specific nodes, per-node routing tables, and rules for updating the table on topology changes. Lookup queries are routed from the starting node in a few hops towards the target node. The routing can be done in O(log N) hops, where N is the number of nodes, due to the small amount of node references managed by a single node (Steinmetz & Wehrle, 2005). Further, DHTs provide load-balancing because the node and data identifiers are distributed nearly equally through the overlay network. Figure 14 depicts an exemplary lookup query in a structured overlay network.

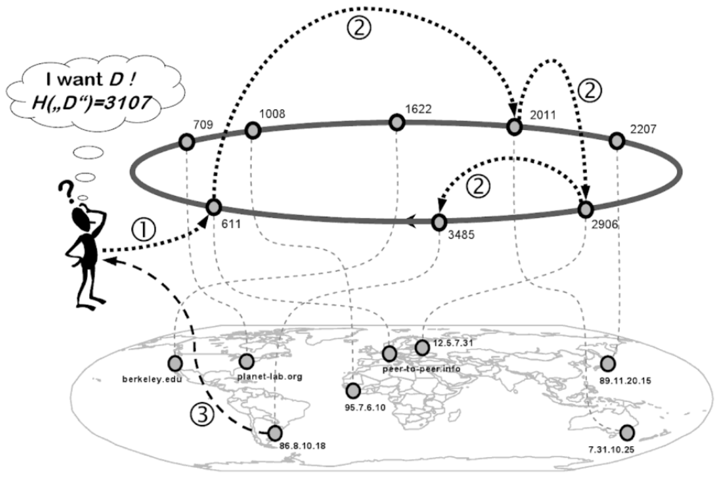


Figure 14: Exemplary DHT lookup query by Steinmetz and Wehrle (2005). (1) Lookup request from node A for item D send to a random node. (2) Request is forwarded in O(log N) hops to target node. (3) Target node sends D to A.

2.3.3 Serverless

With the clarification of cloud computing and the different service models (see Section 2.1), we are now able to discuss the serverless paradigm. Serverless refers to a new generation of PaaS (Adzic & Chatley, 2017). The service provider is responsible for receiving and responding to client requests, task-scheduling, operational monitoring, and capacity planning. Serverless can be a misleading term because servers are still needed to execute functions and host applications. van Eyk, Iosup, Seif, and Thömmes (2017) identify the following three key properties of a serverless architecture:

a) Granular billing: Users are only charged when the application is executed.
b) No operational logic: The infrastructure provider is responsible for operational logic, such as auto-scaling or resource management.
c) Event-driven: Serverless applications respond to events, e.g., display a new instant message.

The question now is: how does serverless fit into the above-explained service models of cloud computing? van Eyk et al. (2017) use the perspective of who manages the operational logic. Figure 15 shows that serverless fits in the gap between PaaS and SaaS, while it marginally overlaps with them. They argue that this overlapping is based on some example services that show serverless characteristics. The main concepts of serverless are backend as a service (BaaS) and FaaS.

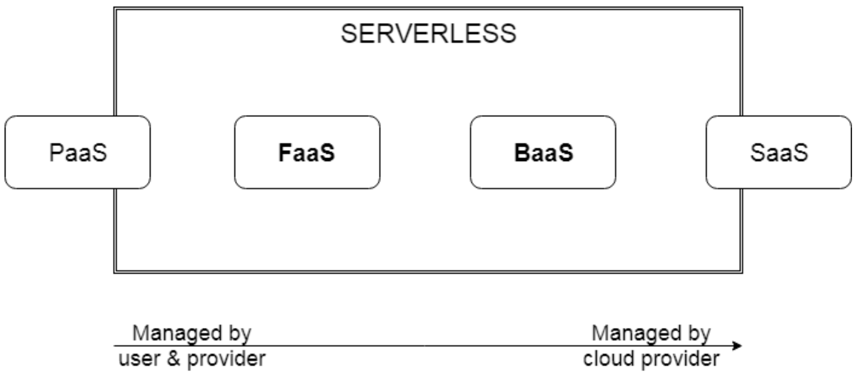


Figure 15: Serverless services overview by van Eyk, Iosup, Seif, and Thömmes (2017)

BaaS provides developers the capability of connecting their applications to backend cloud processing and storage (Lane, 2015). Additionally, other common features, such as user management or push notifications are supported. FaaS focuses on the management of resources, life-cycle, and event-driven execution of user-provided functions by the cloud service provider (van Eyk et al., 2018).

Developers implement small, stateless functions - the cloud provider is then responsible for managing the operational aspects of these functions. As we can see in figure 15, the user is still responsible for managing the operational logic to some extent. Examples include configuration management, such as the number of CPUs, data storage space and amount of service instances.

Figure 16 depicts the most important FaaS providers. All major cloud providers (Amazon, Google, Microsoft, IBM) are offering FaaS. Offered services vary in costs, supported languages, scope, security, and resources. Amazon was the driving force of the shift to cloud computing and provided the first serverless platform: Amazon web services (AWS) Lambda (Amazon, 2014). Users benefit from the huge AWS ecosystem, which facilitates the use of Lambda functions (Baldini et al., 2017). Code executed on Lambda is called Lambda functions. Lambda functions can be triggered by different event sources such as an HTTP request over the Amazon API gateway or mobile apps. There exists also work in the area of open-source serverless computing. OpenLambda is the first defined open-source serverless computing platform (Hendrickson et al., 2016). OpenLambda aims to serve as a playground for exploring new serverless computing techniques. It is written in the programming language Go and operates with Docker containers.

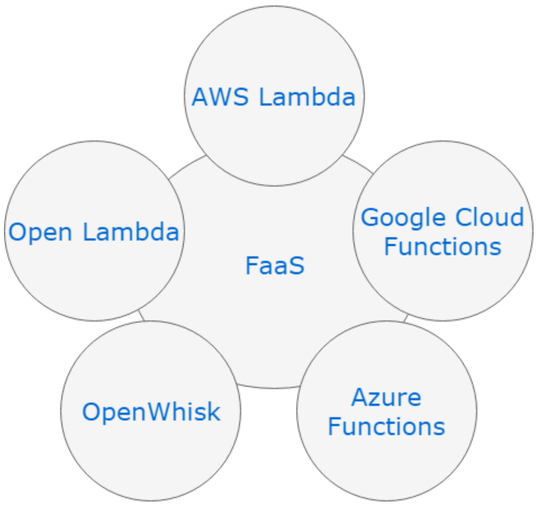


Figure 16: Overview of most important FaaS providers adapted from Fox, Ishakian, Muthusamy, and Slominski (2017)

Serverless architectures can be used in various application areas. According to Fox, Ishakian, Muthusamy, and Slominski (2017), serverless is mostly suitable in short-running, event-driven, and stateless scenarios, such as, bots or micro-services. Serverless is limited on long-running, stateful scenarios, because of the complexity of state maintenance.

2.4 Security Aspects of Instant Messaging

This chapter aims to to discuss IMS security requirements and to give an overview of potential threats that arise on IM usage.

2.4.1 Security Requirements of Instant Messaging

In Chapter 3, we see that IMSs are widely adopted in the workplace. Before introducing a new IMS in the workplace, the following security requirements should be considered (Meletiadou, 2010; Unger et al., 2015):

a) *Confidentiality*: Only intended message recipients can read the message. Confidentiality is usually provided by message encryption, depending on the underlying protocol. There are two encryption schemes used, namely, encryption in transit and end-to-end encryption (De Luca, Das, Ortlieb, Ion, & Laurie, 2016). Encryption in transit means that messages are sent encrypted from the sender to the server and from the server to the recipient. The server can read the contents of the message. End-to-end encryption provides message encryption from the sender till the decryption on the recipient's client without exposing the information to the server or any other third party. The majority of modern IMSs use encryption in transit, mainly because of the reason that end-to-end encryption is more complicated due to its required key exchange between the communication parties.
b) *Integrity*: It must be ensured that sent messages are not altered during the transit by any third party, e.g., an attacker. Any honest party does not accept modified messages. Message integrity is usually verified with a cryptographic hash function.
c) *Authenticity*: Authentication is the process of verifying a claimed identity. In the context of IM, users need to authenticate themselves, usually providing a username, and password chosen on registration.
d) *Privacy*: Patil and Kobsa (2004) conducted a study about privacy concerns in IM. They found three main privacy concerns: privacy from non-contacts, privacy regarding availability, and privacy regarding content. An aimed feature is the ability to control presence appearance.
e) *Availability*: Appropriate countermeasures should be taken to avoid unintended disconnection to an IM server or communication partner. DOS attacks (explained in Section 2.4.2) can lead to server crashes jeopardizing the typical IM process.
f) *Accountability*: The participation in a conversation cannot be denied by any communication party towards a third party. This reduces the level

of privacy. The explicit missing of accountability can also be a desired security requirement (see deniability).

g) *Deniability*: Avoiding deniability or ensuring non-repudiability is a common goal for secure IMS (Unger et al., 2015). For instance, Bob accuses Alice of sending a specific message; a judge has to decide whether Bob is right. If Bob is not able to prove that Alice sent the message, then the action is repudiable or deniable.
h) *Anonymity/Pseudonymity*: Even if the IM traffic is sniffed by a third party, in ideal circumstances, not only the message contents are unreadable, but also the messages cannot be linked to specific users sending or receiving them.

One important aspect that tends to be forgotten is usability. Musiani and Ermoshina (2017) state that there exists a common trade-off between security and usability in IMSs. They argue that most of the public messaging tools do not implement best security practices such as end-to-end encryption. Further, messaging tools that are considered to be secure suffer from usability shortcomings, which may even result in accidentally exposing communication data due to wrong usage, e.g., using too simple passwords or accidentally leaking them. Most of the security requirements are tackled in the underlying IM protocol. See Section 2.5 for more information.

2.4.2 Security Threats of Instant Messaging

In this section, we use the categorization of IM threats into malicious and unintentional threats by Curry (2013). Before discussing the different threats, we also want to explain a model of how a potential attack can be analyzed. The cyber kill chain is a model used by digital forensic investigators and malware analysts to work in a chained manner on an incident response (Yadav & Rao, 2015). Cyber kill chain follows the divide-and-conquer principle by breaking an attack down into multiple layers. While an attacker needs to traverse all layers for a successful attack, defense measures can be applied at all layers to interrupt the cyber kill chain. According to Yadav and Rao (2015), the cyber kill chain consists of 7 layers: 1) Reconnaissance: Information gathering about the potential target. 2) Weaponize: Design of a backdoor and penetration plan by using the information gathered from reconnaissance. 3) Delivery: Attack transmission to target environment. 4) Exploitation: Trigger of an attack on the target system. 5) Installation: Establish a beachhead at the target system. 6) Command and Control: Remote covert instructions to compromised machines. 7) Act on Objective: Causing data exfiltration, network spreading, or system disruption after command execution.

Malicious Threats

Introducing IM in a company opens new security holes (Rittinghouse & Ransome, 2005). IMSs suffer from poor configuration and implementation opening backdoors and potential weaknesses, together with the costs of hosting an own IM infrastructure and application. Like any other internet-enabled application, IM constitutes a potential target for malware (short for malicious software). Malware is any type of programming intended to harm a computer user. Common malware types are viruses, worms, trojan horses, spyware, browser hijackers, or blended threats, which combine the characteristics of multiple distinct malware types. In general, IM can be used by hackers for various malicious purposes (Rittinghouse & Ransome, 2005):

a) As *Carrier*: Malware can be easily distributed in an IMS, because of the robust communication channel between users and their corresponding buddy lists. These buddy lists serve as an address book to automatically spread worms, removing the need to find vulnerable machines.
b) As a *Staging Center*: As explained in Section 2.3.1, the formerly most popular IMSs implemented a client-server architecture. Servers represent a favorable target because hackers gaining control of such servers can impersonate users, eavesdrop conversations, or distribute malware with little effort.
c) As a *Vehicle for General Hacking*: In general, IMSs are similar to other internet-enabled applications in that they may contain exploitable bugs. Hackers could obtain access over user machines, using exploits for vulnerable IM clients.
d) As a *Spy*: Data sent over IM can be sniffed in order to obtain confidential or sensitive information, especially when data is sent unencrypted. Resulting damage, due to unauthorized information disclosure, can exceed damage from direct malicious attacks. There exist several impersonation techniques. The most frequently used attack is stealing account information. Hackers trick unsuspecting users into opening an attachment containing a password-stealing trojan horse. On execution, the trojan horse locates the password on the victims machine (maybe stored in the cache or unobfuscated in the registry), and sends it back to the attacker. People on the buddy list won't notice any difference. A man-in-the-middle attack can hijack IM connections. The hacker can impersonate parties by injecting proper messages in the current chat session. Another impersonation technique is social engineering (Curry, 2013). IM can be used by social engineers to trick users into exposing data with the help of psychological tricks. Therefore, context must be presented to the victim, which appears to be normal. An exemplary scenario is using an identity similar to a boss or colleague and provide information to, in turn, get information.

e) As a *Zombie Machine*: Hackers may aim to use a target machine for their own purposes besides collecting data. The victimized machines can then be used to start, e.g., distributed denial of service (DDoS) attacks. The main idea of DDoS is to flood a target with requests causing slowdowns or crashes.
f) As an *Anonymizer*: Public IMSs allow users to choose arbitrary IDs providing anonymity to some extent. Spoofed enterprise domain names can be used by hackers, for example, to pretend to be an employee of the company. Another vulnerability is SPIM (Spam through IM). SPIM is mainly used by scammers to trick users into opening a URL or a file or sending unwanted advertisements.

Unintentional Threats

"It's human to make mistakes and some of us are more human than others.", while this quote by Ashleigh Brilliant can be considered as common sense, it also applies to the use of IM. Incorrect use of IM can circumvent the taken security measures and thus, lead to threats. Intellectual property leakage of sensitive information, such as source code, insider information, or contracts, might occur when shared via IM. While normally such transmissions can be done safely (with encryption and authentication) over IM, incorrect use might expose those documents to a third party. Moreover, it can also happen that a document is unintentionally sent to the wrong person. In Chapter 3 we outline that IMS in the workplace is also used for non-business related conversations. This is mainly based on the informal nature of IM, together with the use of emoticons and acronyms. If non-business related conversations take prevalence, the risk of inappropriate advances and commentaries is rising. Additionally, users might be distracted from frequent interruptions, which in turn might potentially lead to errors harming the company. In general, companies using IMSs should implement a policy outlining the inappropriate IM use.

2.5 Basic Instant Messaging Protocols

This chapter provides an overview of IM protocols. The open standard IM protocols XMPP and SIP/SIMPLE are examined. The focus lies on discussing only the principles and ideas of those protocols. A detailed discussion is out of the scope of this thesis.

2.5.1 XMPP

The Extensible Messaging and Presence Protocol (XMPP) is a protocol initially defined in the RFC3920 standard (Saint-Andre, 2004) and used *"for streaming XML elements in order to exchange structured information in close to real-time between any two network endpoints"*. It was formalized initially under the name Jabber by the open-source Jabber community in 1999, aiming to provide a secure, spam-free, open, and decentralized IMS. Most of the XMPP implementations follow a strict client/server model (Saint-Andre, 2005), requiring the data to traverse at least one server towards its destination. Figure 17 shows the essential XMPP components: XMPP client, XMPP server, and gateways to foreign networks. While the server is responsible for managing connections and routing messages, gateways are needed for connecting different networks. TCP is used as the underlying connection protocol. Before initiating a new session, we first need to clarify how a client is identified.

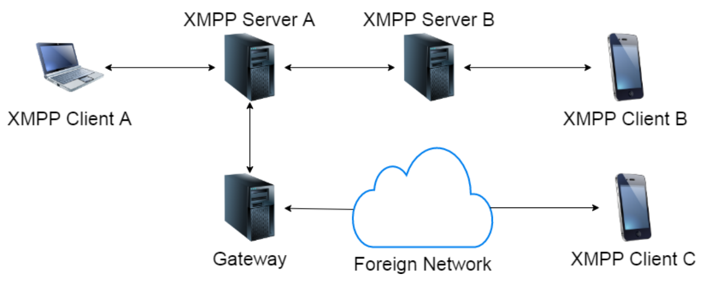


Figure 17: XMPP architecture

Every XMPP entity is identified by an address called JabberID (JID) (Saint-Andre, Smith, & TronCon, 2009). JIDs use a syntax similar to email: *username@domain*, where the domain contains an installed XMPP server, e.g., *lukas@siemens.com*. The username is unique only for a certain domain, meaning that the same username can be used several times on different domains. XMPP allows simultaneous usage of multiple devices. Therefore, the JID is extended with a resource identifier, enabling exchanging messages with specific devices: *username@domain/resource*. In order to prevent any message delivery issues for multiple available devices, devices can be prioritized. If the resource part is missing, the device with the highest priority receives the message first. If two devices with the same priority are available, the device that signed on, at last, receives the message first. The basic XMPP concept is XML streaming using specific XML elements (so-called XML stanzas) (Saint-Andre, 2004). An XML stream is a container used for exchanging XML elements between two entities over a network. The number of XML elements is not restricted. XMPP uses an opened `<stream/>` tag to indicate the start and its corresponding closing tag for the end of an XML stream. An XML stanza is a unit of structured information sent over an XML stream. It is located as a child directly under the aforementioned stream tag. A new client-server connection results in two opened XML streams (one from the client to the server and vice-versa). After stream parameter negotiation, any communication party can start sending XML stanzas. There are three core stanza types:

1) `<message/>`: Used to send a message from one entity to another. It provides several attributes: type (used for example to indicate if its chat or group chat), from and to containing the corresponding JIDs and the basic attributes body and subject to define the payload.
2) `<presence/>`: Used as a general publish-subscribe mechanism to distribute presence information. Only subscribed entities receive presence information. This enhances privacy because only selected parties can see this information. The attributes show and status are available to indicate presence and a user-specified status message.
3) `<iq/>`: Used as a general request-response mechanism similar to HTTP requests. The type attribute is used for example, by a client to request a resource and the server to respond with a result.

In listing 2, an exemplary XML stream snippet is illustrated. On lines 1-5, a client requests his stored contact list. The server responds with two saved contacts (lines 7-12). After that, the presence information from a specific contact is following (lines 14-17, “xa” means extended away). Finally, a message is sent from one entity to another (lines 19-22).

```
1  <stream:stream>
2
3      <iq type="get">
4          <query xmlns="jabber:iq:roster"/>
5      </iq>
6
7      <iq type="result">
8          <query xmlns="jabber:iq:roster">
9              <item jid="markus@siemens.com"/>
10             <item jid="philipp@siemens.com"/>
11         </query>
12     </iq>
13
```

```xml
    <presence from="markus@siemens.com">
        <show>xa</show>
        <status>currently in a meeting</status>
    </presence>

    <message from="lukas@siemens.com/smartphone" to="markus@siemens.com"
        type="chat">
        <body>Do you have time for a coffee break after your meeting?</body>
        <subject>Coffee break</subject>
    </message>

</stream:stream>
```

Listing 2: XMPP XML stream example

XMPP aims to provide three basic security requirements: confidentiality, data integrity, and mutual authentication. In order to achieve this, XMPP is built upon two other protocols: Transport Layer Security (TLS) and Simple Authentication and Security Layer (SASL). TLS is responsible for ensuring confidentiality and integrity of exchanged messages. Hence, TLS supports the protection against eavesdropping, password sniffing, man-in-the-middle attacks and stanza replays. SASL can be seen as a framework providing an abstraction layer between certain protocols such as XMPP and mechanisms ensuring authenticity and integrity (Melnikov & Zeilenga, 2006). Currently, it supports the following authentication methods: “DIGEST-MD5”, “CRAM-MD5”, "GSSAPI", “PLAIN”, “ANONYMOUS” and “SCRAM” (Saint-Andre et al., 2009). The XMPP Standards Foundation (XSF) is responsible for standardizing XMPP extensions. Several extensions aiming to increase security, have been proposed or standardized over the last years:

a) *Off-the-Record (OTR)*: The OTR protocol is a cryptographic protocol providing end-to-end encryption for P2P instant messaging (Borisov, Goldberg, & Brewer, 2004). OTR uses 128 bit AES symmetric-key encryption and the SHA-1 hash function. OTR cannot be used for multiuser chat, because a session can only be held between two parties. In contrast to Open Pretty Good Privacy (OpenPGP), it also provides forward secrecy.
b) *OpenPGP*: OpenPGP is a standardized message protocol used to sign and encrypt data exchanged between multiple parties (Callas, Donnerhacke, Finney, & Thayer, 1998). It is mainly used for email encryption. The corresponding XMPP extension XEP-0373 (XMPP Standards Foundation, 2018) specifies the foundations of end-to-end encryption and authentication with the help of OpenPGP. The problem with this extension lies in its key management complexity, making it difficult for users to apply properly. This extension is still under active development.

c) *Secure/Multipurpose Internet Mail Extensions (S/MIME)*: S/MIME is based on MIME, which defines the format of emails (Ramsdell & Turner, 2019). S/MIME provides authentication, privacy, integrity, and non-repudiation. The key management involves a public key infrastructure instead of a decentralized web of trust as in OpenPGP (Ermoshina, Musiani, & Halpin, 2016).

2.5.2 SIP/SIMPLE

Session initiation protocol (SIP) is an application-layer protocol used for establishing sessions in networks and is defined in RFC 3261 (Rosenberg et al., 2002). SIP aims to provide multimedia sessions (conferences) such as voice-over-IP. SIP consists of the following network components (Cumming, 2003): a) User Agent (UA): UA refers to a SIP endpoint. It can either act as a client or as a server. b) Proxy: Devices that are responsible for routing data to its destination. c) Registrar: Specialized server UA responsible for handling REGISTER requests (explained in the next paragraph).

A SIP entity is identified by a SIP Uniform Resource Identifier (URI) using the syntax: *sip:user@*[*domain|host|ip-address*]. The communication between UAs is realized using requests and responses, similar to HTTP requests. SIP requests are distinguished based on their method. Some basic methods are REGISTER, used for UA registration at a SIP proxy and INVITE, used for connection establishment. Figure 18 illustrates how a connection between the UAs Bob and Alice is established:

1) Both UAs register themselves at the registrar
2) Bob requests a new session with Alice by sending an invite request
3) The proxy forwards the request to Alice.
4) Alice accepts the new session by sending a "200 OK" response.
5) The proxy forwards the response to Bob.
6) A new session between Alice and Bob is established.

SIP for IM and presence leverage extension (SIMPLE) is an extension for the SIP protocol to support IM and presence functionality and is defined in RFC 3428 (Campbell et al., 2002). Therefore, new request methods were introduced:

a) SUBSCRIBE: This method is used by a UA to subscribe to presence information of another UA. The corresponding request contains the SIP URI of the aimed presentity.
b) NOTIFY: Subscribed UAs receive such requests containing the presence information of a presentity.

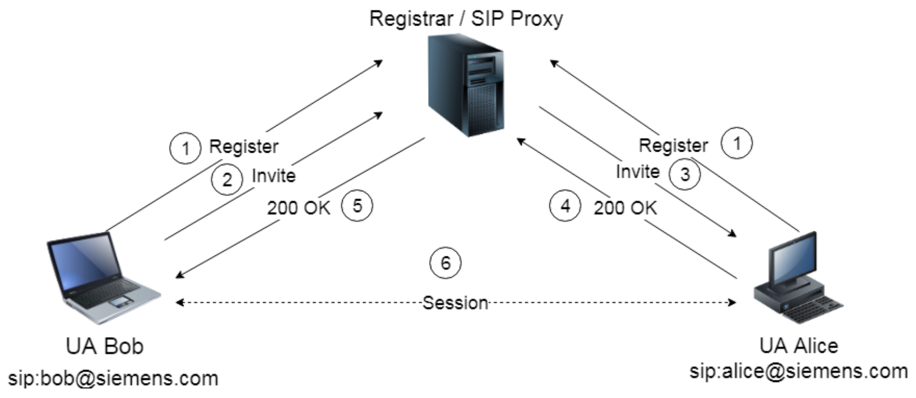


Figure 18: Simplified SIP session establishment

c) MESSAGE: The actual IM is carried out via these requests. These requests contain the actual message content in MIME format together with the SIP URI of the target inbox.

Figures 19 and 20 show the typical sequences of exchanging presence information and messages.

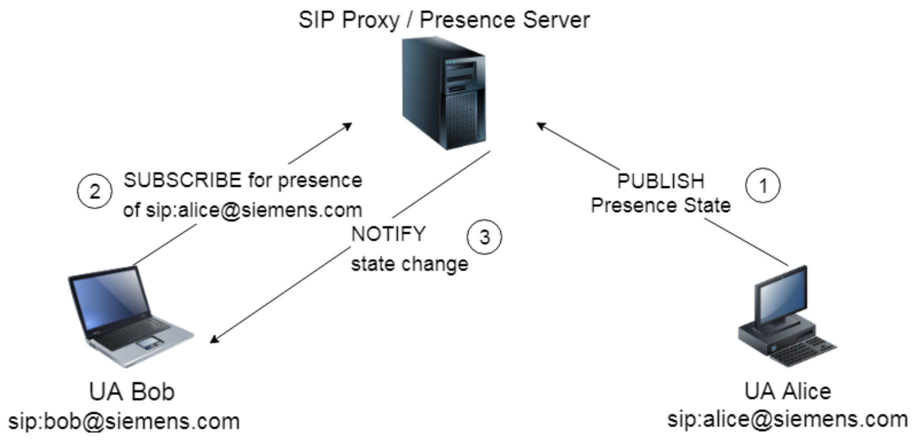


Figure 19: SIMPLE presence information exchange

There exist two different IM modes: page and session mode (Rosenberg, 2013). The page mode sends messages without establishing a session. This means that every message is routed independently to its destination. While this mode is efficient for a small number of messages, it is not suitable for conversations involving more data, because all data is sent over the SIP serves and not directly. On the contrary, the session mode (as the name implies) establishes a session between the communication parties. The RFC standard 3261 (Rosenberg et al., 2002) describes the following security mechanisms provided by SIP. SIP security is realized either in a hop-by-hop or end-to-end fashion. Confidentiality, integrity, and data origin authentication is achievable by using

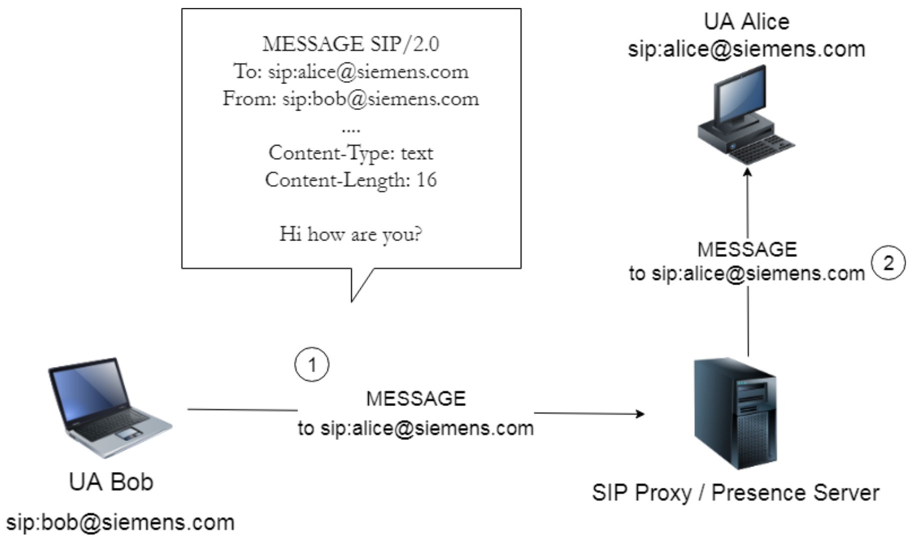


Figure 20: SIMPLE message exchange

TLS and IPSec (Geneiatakis, Kambourakis, Dagiuklas, Lambrinoudakis, & Gritzalis, 2005). SIP Secure is a mechanism to provide end-to-end protection, similar to HTTPS, causing the SIP URI prefix to change to *sips*. Similar to XMPP, SIP also uses S/MIME to send encrypted and authenticated messages.

2.6 Real-Time Communication

IM, online gaming, video conferencing, and many other applications need real-time communication (RTC). RTC refers to any kind of communication, where users exchange information with guaranteed or predictable latency. In the context of this thesis, IM with real-time communication is essential because users need to be updated in nearly real-time. Other types of CMC, such as e-mail or forum, do not represent an alternative to IM because they are not real-time capable.

There are two real-time communication modes: half- and full-duplex. Half-duplex means that a user cannot send and receive at the same time, whereas messages can be send and received simultaneously in a full-duplex mode. There exist various RTC enabling technologies. According to Liu and Sun (2012), polling, long polling, and HTTP streaming are the main technologies enabling real-time communication used in the past, and WebSockets is the current state-of-the-art.

Polling

Polling describes a mechanism, where the client requests periodically data from the server. The advantage of this approach lies in its easy implementation. The main drawback of this approach is that the server cannot push new data to the client on demand. This causes unnecessary network traffic, as new data might not be available while requests are always sent.

Long-Polling

To overcome the shortcomings of polling, a new approach, called long-polling, was introduced. The client polls new information from the server, and the server keeps the request open until new data is available. Thus, the use of network resources and the client-server message delivery traffic is minimized (Loreto, Saint-Andre, Salsano, & Wilkins, 2011). Usually, after a long-polling request has been processed, the client immediately opens a new one. Polling and long-polling are criticized for frequent opening and closing of TCP/IP connections.

HTTP streaming

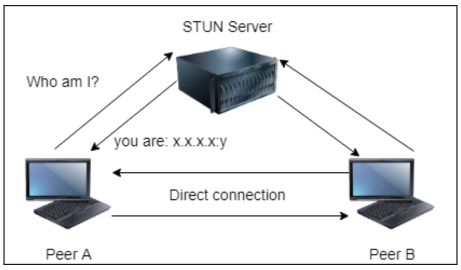
Another approach is HTTP streaming. The difference to the approaches mentioned above, is that an opened HTTP connection remains open indefinitely. A server is capable of sending information segments within the same response without the need for connection termination. Network latency is reduced by the absence of multiple connection establishment. Since connection completion signaling is not done, server responses can get buffered by network intermediaries, such as firewalls, resulting in potential data loss or errors.

WebSockets

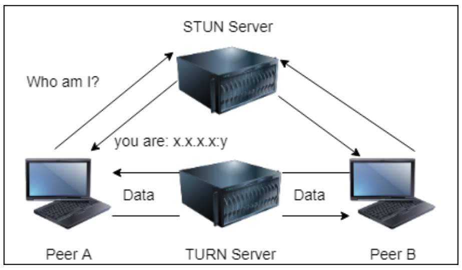
WebSockets were designed to address the shortcomings of the described mechanisms. The WebSocket protocol defines a full-duplex, real-time communication channel over a TCP connection (Fette & Melnikov, 2011). A WebSocket channel is established by using an HTTP request, asking the other communication party for a connection upgrade. When the request is accepted, subsequent messages are sent via the WebSocket protocol. Once a connection is established, the communication parties can send data asynchronously to each other. Connection closing is only performed, if one of the communication parties actively closes it. Thus, network traffic and latency are significantly reduced.

2.6.1 WebRTC

Web real-time communication (WebRTC) is a collection of standards and protocols for enabling audio/video calling, video chats, and P2P chat for browser-to-browser applications, without needing any third-party software (Sergiienko, 2014). WebRTC is open-source and was published by Google in 2011. It is still undergoing active development. The fundament of WebRTC are components that can be easily accessed and used via JavaScript and HTML5 in browsers. This removes the need for users to install or manually configure any browser plugin. Another advantage is that needed multimedia functions, such as codecs or stream management, are already integrated into the browser. Connection establishment does not work without a signaling server. Any communication mechanism, for instance, WebSockets, allowing the exchange of session description protocol (SDP) data, can be used for signaling (Sergiienko, 2014). SDP is used to describe multimedia sessions in terms of session invitation, session announcement, and parameter negotiation (Levin & Camarillo, 2006). Connection establishment is more complicated when the communication parties are behind firewalls or use network address translation (NAT). Two basic concepts are supporting these scenarios: session traversal utilities for NAT (STUN) or traversal using relays around NAT (TURN). Figure 21 depicts the general idea of STUN and TURN. STUN servers aim to solve the NAT/firewall traversal issue in order to find the other peer. After the connection is established, data is directly transferred between the peers. On the contrary, TURN servers are used when STUN servers are not enough to perform successful signaling. An example will be if both peers are behind a symmetric NAT. TURN servers act as some kind of re-transmitter between peers, resulting in data traversal through the specific TURN server.



(a) STUN server



(b) TURN server

Figure 21: STUN and TURN server concept adapted from Sergiienko (2014)

easyRTC

easyRTC is an open-source JavaScript library (Priologic Software Inc, 2019) that supports building WebRTC applications. Integration into an HTML page is done similar to any other JavaScript library. An already implemented signaling server based on Node.js manages signaling. The signaling server provides STUN and TURN support. In order to provide a reliable connection, easyRTC is equipped with a fall-back strategy to WebSockets, when connection issues occur.

2.6.2 SignalR

SignalR is a framework from Microsoft targeting ASP.NET that is used to integrate RTC in web applications. Several Microsoft Office tools use SignalR to enable real-time collaboration features (Aguilar, 2014). SignalR facilitates message delivery in real-time between peers by providing an abstraction layer hiding low-level details. Further, it supports several connection mechanisms (WebSockets or long polling) to open a (virtual) persistent connection. One important concept is *hubs*. Hubs are the API used to access the created persistent connection. They can be seen as a two-way RPC, as hub methods are invoked from the client and the server. Client-side code is called from a hub by sending messages containing the corresponding method's name and parameters. The client then matches the name to an existing method and executes it (if matching was successful). A client SDK, such as .NET or Java, is used to connect to the dedicated SignalR hub. SignalR is self-hostable or can also be used via the Azure SignalR service to obtain a managed platform.

2.7 Graphical User Interface Design

According to Galitz (2007), the user interface (UI) is the most important part of a computer system because it appears to be the whole system for users. Responses to poor UI design range from psychological responses, such as frustration, panic or stress, to physical responses, such as only partial system usage or even to abandonment of the whole system. To avoid such responses, it is essential to consider the design process itself. Any UI design process should start with an understanding of the system's users. This is an important, yet often underestimated task. Without the understanding of the system's user, the system might not meet the user requirements. So the goal is to overcome the gap in knowledge, skills and behaviors of the system users and the developers. Cooper, Reimann, and Cronin (2007) argue that only qualitative research techniques can achieve this.

Qualitative research techniques include stakeholder interviews, literature review, prototype audits, or user observation. The next step after research is the creation of descriptive user models. These models are often referred to as personas. Personas are a composite archetype or fictional character representing a specific user type. Thus, personas do not only help in understanding the user's way of thinking, they also support in resolving the following three design issues: the elastic user, self-referential design, and edge cases. Elastic user refers to the phenomenon of tuning the final product based on the opinions of anybody giving input while losing sight of the real user's requirements. When developers use their own mental models, goals or skills, the self-referential design issue could arise. The final product might look perfect in the perception of the developers but not of the customer. Personas put the main focus on the requirements of the real users, thus they also help to avoid designing only for edge cases. While personas represent individual human beings, a base goal in UI design is to address the needs of different user experience levels with a single interface (Cooper et al., 2007). User experience levels are beginners, intermediates, and experts, while intermediates refer to the largest group of users. Over time both the beginners and experts tend to drift towards intermediates. Beginners aim to leave the beginner state fast to reach the expert level. While their learning curve is steep in the beginning, it flattens down fast, resulting in the intermediate level. Experts can forget their knowledge over time and, thus fall back into the intermediate level. A well-designed UI needs to balance the requirements of the different user experience levels. An intuitive and comfortable UI helps the user to become quickly familiar with the system, increasing its overall system acceptance. Figure 22 illustrates the different user experience levels and their corresponding design requirements. We see that the requirements vary with each level. Cooper et al. (2007) argues that the main goal is *"to rapidly and painlessly get beginners into intermediacy, to avoid putting obstacles in the* *way of those intermediates who want to become experts, and most of all, to keep perpetual intermediates happy as they stay firmly in the middle of the skill spectrum".*

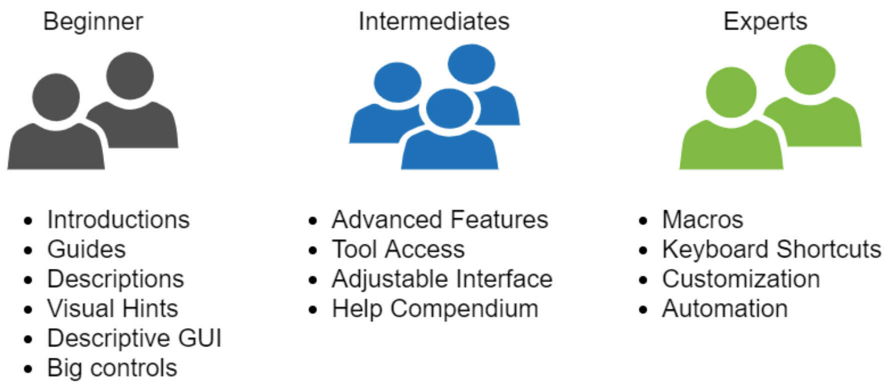


Figure 22: Design requirements of different user experience levels (Cooper, Reimann, & Cronin, 2007).

Tremendous effort during the development process will be put on usability. Galitz (2007) describes usability as *"a quality attribute that assesses how easy a user interface is to use."* It is one of the most important qualities of a UI. Five components can describe usability: learnability, efficiency, memorability, amount and severity of errors, and satisfaction (Nielsen, 2012). While this sounds easy in theory, it is very hard to assess or measure usability in practice. There exists the concept of usability or user testing, which is a collection of techniques measuring usability in terms of user interaction with the system under test (Galitz, 2007). The main idea of usability testing is the measurement of how well-standardized tasks are resolved and which errors occur while task solving. Representative users should be chosen for this task, e.g., a customer. Usability professionals often track the users with audio or video recorders in order to better retrace the user's behavior afterward.

3 Related Work on Instant Messaging

There exists a significant amount of work in the area of IM. Numerous IMSs have been implemented over the past decades. Driven by the increased popularity of smart-phones, many mobile IM applications emerged. This caused the discontinuation of some popular IMSs, e.g., the MSN Messenger. In this section, the main focus is on the use and impact of IM in the workplace rather than discussing different IMSs. The majority of studies support the presumption that the positive work-related effects of IM outscore the drawbacks, such as frequent interruptions or distraction. In many studies, the integration of an IMS is considered as controversial.

3.1 Positive Work-Related IM Effects

Garrett and Danziger (2007) investigated the influence of IM on the workplace. Results show that IM reduces interruptions and encourages communication. They argue that this happens because *"workers are using IM technology to manage interruptions, postponing work-related communications until they are more relevant or less disruptive."* Additionally, IM offers an efficient communication and information exchange mode that improves colleague intercommunication. Wu, Liang, Chiu, and Yuan (2017) explored the impact of IM use on employee empowerment. The study provides three main findings: 1) IM has a positive impact on employee empowerment. 2) Employee empowerment has a positive effect on the organizational commitment of employees. 3) IM use intensity does not correlate with job satisfaction.

Isaacs, Walendowski, Whittaker, Schiano, and Kamm (2002) investigated many logged IM conversations of employees. The conversations are evaluated under conversational characteristics. The outcome illustrates that the primary IM use is for complex work discussions, such as problem-solving or social learning, rather than asking simple questions or distributing information. Additionally, they showed that people rarely switch from IM to another medium when the conversation gets complex. This contradicts the common perception that IM is used for simple discussions and that the more complex discussions are carried out in person. Dittrich and Giuffrida (2011) explored the role of IM for a co-located software development team. Their analysis indicates that IM fulfills a special role: it supports trust-building and social relationships of co-workers by acting as real-time glue between different communication channels. A study by Ou, Davison, and Leung (2014) based on 41 survey participants from a small-sized company in Hong-Kong shows that the use of IM is highly correlated with knowledge generation. They further argue that, because of this correlation, IM, improves work performance. Salovaara and Tuunainen (2013) conducted a case-study on a 150-person software company, where the developers use Skype chat as their favorite knowledge sharing medium. In their findings, they describe that IM can be a powerful knowledge sharing tool for ephemeral project-based knowledge. Lebbon and Sigurjónsson (2016) examined whether frequent interruptions due to IM could negatively affect performance. They suggest that IM leads to higher productivity, even if IMSs are also used for non-work related communication. Pi, Liu, Chen, and Li (2008) used a social influence model to show that IMSs can improve communication efficiency and, thus reduce communication costs in enterprise organizations. The communication satisfaction of employees also increases with the use of an IMS. Jaanu, Paasivaara, and Lassenius (2012) conducted a study about the use of IM in distributed software engineering projects. Media synchronicity theory was used to prove that IM is best suited for simple discussions. Software engineers prefer other media, e.g., teleconferencing, for complex discussions. They further argue that for maintaining communication efficiently, a combination of IM and other communication media is needed.

Niinimäki and Lassenius (2008) interviewed 39 software developers in their study on the use of IM in their global software development projects. In all successful projects, IM was used systematically. Practitioners argued that IM helped in multitasking and maintaining communication with multiple people. Additionally, IM offers a lower communication initiation barrier compared to other communication media. Hönlinger (2018) presents results from a practitioner survey with 176 responses. Respondents were asked to describe the impact of IM in Germany on teamwork performance and knowledge sharing. The highest ratings were given for the positive influence on knowledge sharing and teamwork performance improvement.

Rennecker, Dennis, and Hansen (2006) conducted a study about how IM can be used to restructure meeting boundaries in face-to-face and technology-mediated meetings. They used Goffman's characterization of front and back-stage interaction practices. For this purpose, they interviewed 22 managers and workers in U.S. based organizations. Results show that IM is used to participate concurrently in front and several backstage interactions. They argue that without IM, these interactions are physically impossible.

3.2 Negative and Controversial Work-Related IM Effects

Gupta, Li, and Sharda (2013) examined the consequences of IM on task quality and duration. They suggest that with the use of IM primary task quality is decreased. Task duration highly depends on the hierarchical level of the message sender. Messages sent from, e.g., a supervisor, decrease task duration while simultaneously decreasing task quality. Messages from other peers increase task duration. Mansi and Levy (2013) compared the task completion time of knowledge workers for different task complexities with frequent IM interruptions. Results show that IM interruptions do not affect the task completion accuracy but significantly increases the task completion time. Li, Gupta, Luo, and Warkentin (2011) studied IM influences on multitasking satisfaction and perceived task complexity. They suggest that satisfaction with multitasking highly depends on the individual polychronicity. Thus, frequent interruptions decrease the satisfaction of monochronic employees.

A study by Czerwinski, Cutrell, and Horvitz (2000b) investigates the effects of IM notifications on currently performed tasks. Results show that the disruptiveness of IM highly depends on the point of time during task computation. Interruptions in early phases are considered to be not as harmful as interruptions when a user is deeper engaged with a task. Further, they argue that IM disruptiveness is low when incoming messages are highly task relevant. Building on the previous study, Czerwinski et al. (2000a) analyzed in another study the influence of IM on the performance of different searching task types. They conclude that interruptions are more harmful to fast, stimulus-driven search tasks rather than effortful semantic-based search tasks.

Ou and Davison (2010) conducted an empirical study about the impact of IM in the workplace based on 253 survey participants in China. They describe IM as a *double-edged sword* because it concurrently provides several drawbacks and benefits. Overall, they argue that the negative effects are negligible, due to the significant increase in communication performance. Quan-Haase (2010) investigated the negative effects of IM within the workplace. The findings show that IM is disruptive and can even lead to a decrease in productivity. She further argues that with the creation of routines and self-regulation, the negative effects can be decreased.

Rennecker and Godwin (2003) investigate the unintended consequences of IM for worker productivity. Although IM supports certain tasks and decision processes, the productivity suffers from unstructured IM. This origins from the increase of communicative workloads, participation in multiple conversations, and interruption frequency.

Fussell, Kiesler, Setlock, and Scupelli (2004) analyzed the effects of IM on the management of multiple project trajectories. They argue, that IM user interfaces need to support the following features for supporting project trajectory management: i) An awareness component providing availability information. ii) An information component providing an indication when collaborators are working on a task. iii) A reminder component providing a short to-do description of tasks and activities.

4 Problem Statement

This chapter explains the underlying problem description and the research questions that are answered within this thesis. Further, the requirement sdefined by Siemens for the IMS are presented.

4.1 Problem Description

Rising complexity in industrial manufacturing requires companies to develop efficient engineering processes for accommodating individual customer requests. Today, machine builders, plant operators, and system integrators do not need to work co-located on the same machines and plants anymore. They even can be distributed on different countries or different companies or subcontractors. In the context of Siemens, their customers use the same automation software to program on programmable logic controllers (PLCs) and access human-machine-interfaces (HMIs) or motion control devices. Several problems arise due to the absence of an integrated communication mechanism within the automation software. While engineers can be from different companies, they also might use different IMSs depending on their company policies. There is no unified or established IM process. Usually, they need to agree upon a common communication mechanism, however, in practice we have seen that a conglomeration of tools like Skype, Whatsapp, Email, etc, are used. This does not only lead to communication overhead but also can constitute a security vulnerability if any insecure messaging tools are used. Another problem arises from the automation process itself. For instance, if an engineer needs to apply changes like a firmware update or alter the code that is executed on a PLC, the necessity of notifying other engineers currently programming on the corresponding PLC arises, as the PLC will not be accessible in that time. While the process of downloading new content to the PLC can take several minutes, the engineers do not know when the PLC will be available again or which changes have been made. The information-gathering process might suffer from misconceptions about the originator, resulting in potential misunderstandings or data loss.

The goal of this thesis is to implement and integrate a messaging system in the automation software aiming the resolve the aforementioned problems. The chosen approach aims to follow the zero-configuration scheme, while simultaneously considering the limitations and the existing framework of the underlying automation software (described in Chapter 6). Within the scope of this thesis, the following questions are answered:

1. Which start-of-the-art IMS technologies exist?
2. How to enable communication among distributed engineers to avoid misunderstanding, confusion, and downtimes in a controllable unified manner, using existing infrastructures?
3. Which alternative mechanisms exist that could be used for messaging? Is it possible to communicate via connected PLCs?

4.2 Requirements

Several requirements of the IMS have been identified. They are categorized into functional and non-functional requirements. Functional requirements are shown in table 1, and the non-functional requirements are shown in table 2.

RQID	**Name**	**Description**	**Priority**
1	Send message	Functionality of sending a message.	high
2	Presence list	The presence status of other users should be visible.	medium
3	History	Chat messages should still be available after re-connection.	medium
4	Group Chat	Functionality to chat in a group.	medium
5	Notification	Users should get notified on new messages.	medium
6	Message status	Indication whether a message was send/delivered/seen.	low
7	Message timestamp	Display when a message was send.	low
8	Emoji support	The functionality of sending a message containing emojis.	low

Table 1: Functional requirements

RQID	**Name**	**Description**	**Priority**
9	TIA portal integration	The IMS must be available within the TIA portal.	high
10	Costs	No additional costs for Siemens.	high
11	Privacy	Privacy from non-contacts, availability privacy and content privacy.	high
12	Extensibility	The IMS can easily be extended with new features	high
13	Message Confidentiality	Messages should be sent encrypted.	high
14	Zero-config	No complex configuration is needed.	medium
15	Availability	The IMS should provide high availability.	medium
16	Maintainability	The IMS should be maintainable with low effort.	medium
17	Scalability	Number of supported users.	low

Table 2: Non-functional requirements

5 Related Industrial Standards

The target enterprise industry automation software is the Totally Integrated Automation Portal (TIA Portal), developed by Siemens. TIA Portal focuses on the application area of industrial factory automation. TIA Portal consists of several tools, components, and services to integrate all levels of the automation pyramid shown in figure 23. Thus, TIA Portal aims to support machine builders, system integrators, or original equipment manufacturers (OEMs) in simplification and cost savings along the value chain. There exist various norms and standards regarding factory automation. In the following, only the most important norms are explained.

5.1 Standard ISA 95

In general, the model used to describe industrial automation is the automation pyramid defined in standard ISA 95 by the International Society of Automation (ISA) (ANSI/ISA-95.00.03-2005, 2005). The model aims to separate the different manufacturing operations into different interconnected levels, while simultaneously showing the level-specific used information, system, and time-frame types (Åkerman, 2018). The model is depicted in figure 23. On the bottom of the pyramid are sensors and signals which provide a fast and easy data gathering process. They are controlled by the sensing and manipulating level containing PLCs. Level 2 contains supervisory control and data acquisition (SCADA) systems and HMI devices used to monitor devices of underlying levels and their communication. Level 3 controls manufacturing operations and their execution order by a manufacturing execution system (MES). The top-level serves the management to facilitate planning and logistics and is often managed in an enterprise resource planning (ERP) system. Due to the recent shift to cloud computing and its related technologies, we can argue that another level - the cloud level - could be introduced on top of the pyramid. With the help of the cloud, certain automation information is made available via the cloud. For instance, Siemens introduced an internet-of-things (IoT) operating system, called Mindsphere, connecting products, plants, and machines to benefit from the richness of available data.

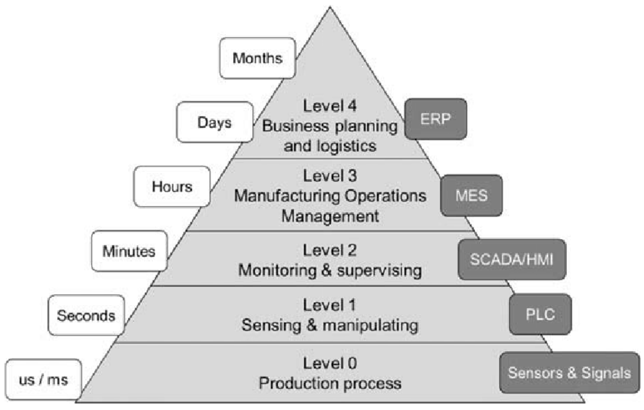


Figure 23: Automation pyramid by Åkerman (2018)

5.2 Standard ISA 88

ISA 88 is a standard that defines models and terminology for describing batch process control. The models provide a classification for machines involved in the batch process and recipes, describing the manufacturing process to ensure standardized batch process automation (Scholten, 2007). In practice, ISA 88 and ISA 95 are used together: ISA 88 for control automation of machines and ISA 95 for information exchange between ERP and MES systems.

5.3 Standard IEC 61131

IEC 61131 is a standard focusing on the basics of PLCs (John & Tiegelkamp, 2009). Within IEC 61131-3, the following programming languages are defined: instruction list (IL), ladder diagram (LD), function block diagram (FBD), sequential function chart (SFC), and structured text (ST). TIA Portal users can select any of them.

5.4 Standard IEC 62264

IEC 62264 is a standard based on ISA 95 and targets system integration of enterprise control systems. IEC 62264 describes level three of the automation pyramid shown in figure 23, by defining activities and interfaces between an enterprise's business system and its manufacturing system.

5.5 Standard IEC 61508

Another important standard is IEC 61508, which focuses on the functional safety of electrical, electronic, and programmable electronic (E/E/PE) safety-related systems. Bell (2006) defines functional safety as *"a part of the overall safety that depends on a system or equipment operating correctly in response to its inputs."* IEC 61508 concerns the impact on the safety of persons or the environment by failures of E/E/PE systems. IEC 61508 uses the safety integrity level (SIL) to quantify the level of needed risk reduction. The qualitative risk assessment evaluates factors, such as the seriousness of the possible harm or injury or the probability of the occurrence of a hazardous event, to calculate the SIL level. In order to reduce the risk of errors, the safety-related construction principles result from the aimed SIL level. Software written in accordance with IEC 61508 needs module tests and suggests code coverage value depending on the SIL level. There exist several sub-standards for industry-specific variants. For instance, IEC 62061 focuses on the safety of machinery, and IEC 61511 targets the process industry sector.

6 Industrial Environment

The target enterprise industry automation software is the TIA Portal, developed by Siemens. TIA Portal is built with the .NET framework and around 1000 developers distributed on several countries and continents implemented for over one decade on it. It is still in active development. TIA Portal targets the Windows platform and can be installed on Windows 7, 10, and Server 2012/2016. Due to its size, we can find various kinds of design patterns ranging from creational, structural, behavioral to concurrency patterns. Examples include the patterns defined by the "Gang of Four" (Gamma, Helm, Johnson, & Vlissides, 1995): command, publish-subscribe, adapter, singleton, mediator, model-view-presenter and dependency injection patterns. The GUI is implemented by custom controls that inherit from the standard windows form controls. The look and feel of these controls are part of the corporate design. Currently, there exists no state-of-the-art control for displaying web content. XML configuration files are extensively used to access and configure standard components. For instance, a toolbar within a view or menu entries does not need to be re-implemented every time, they are simply configured via an XML file. Due to the broad feature scope, the availability of shortcuts is limited. This hinders the setting of intuitive shortcuts for new features.

TIA Portal provides access to a range of digitalized automation services, from digital planning and integrated engineering to transparent operation. One fundamental feature is programming on Siemens Simatic PLCs. PLCs are used to control processes, such as a press for plastic-shaped parts or a robot-gripper. The process occurs according to instructions of the PLC's in-memory program. Actuators are wired to designated outputs of a PLC, allowing the PLC to switch on and off motors or lamps and many other actions. The standard workflow is that clients implement their PLC program, afterwards, they compile their changes to check for potential errors, and finally, they download their program to the dedicated PLC.

Simatic PLCs are endowed with a so-called online-object-model, which aims to make certain automation system information available to the outside, where it is accessed by clients. In order to access such information the specific PLC must be connected to the network with an interface card. The following network types are supported by interface cards: ethernet, process field bus (Profibus), and multi point interface (MPI). Further, the client needs to connect to the corresponding PLC via an integrated TIA Portal mechanism.

Figure 24 illustrates an exemplary automation setup. Three engineers use the TIA Portal to connect to a dedicated PLC via profinet. The PLC is also connected to two IO devices and an HMI panel via profinet and another IO device via Profibus. This setup can be used, for instance for monitoring diagnostic data: The IO devices send their diagnostic data to the PLC. Afterwards, the data is processed, evaluated and graphically displayed on the HMI panel.

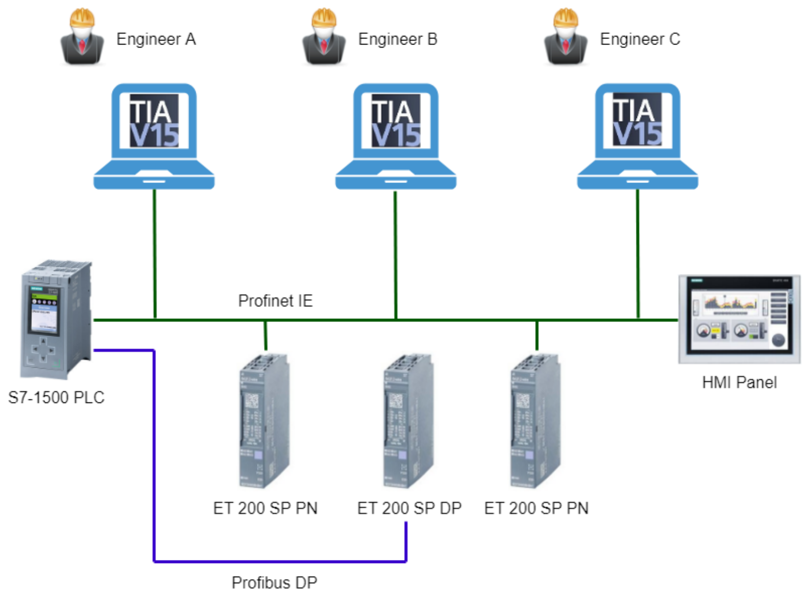


Figure 24: Exemplary automation setup

7 Approach

Within the scope of this thesis, three different approaches have been implemented and evaluated. This chapter aims to provide an overview and explanation of the different approaches. One important aspect is that the goal of this thesis is not to develop a whole new IMS but rather to analyze possible technologies and verify how they can be integrated into the TIA Portal.

- Approach 1 aims to enable IM by integrating a chromium browser control and using the open WebRTC standard for real-time communication. Connection establishment is carried out by a signaling server implemented in Node.js. The main drawback of this approach is the additional NAT handling needed to enable signaling in different NAT configuration scenarios.
- Approach 2 focuses on a cloud-based solution based on .NET SignalR. The IMS is implemented as an ASP.NET Core application and runs on the Azure cloud. Benefits are high scalability, reliability and the absence of server maintenance due to the Azure services usage. The main drawback of this approach are involved costs.
- Approach 3 uses PLCs to transfer messages between connected users. For this purpose, existing TIA Portal services are used to save messages on a PLC. Hence, no third party components are needed. The main drawback is availability, because users need to be connected to a dedicated PLC, which is not possible in certain scenarios.

7.1 Approach 1: WebRTC with Node.js Server

The main idea of the first approach is to use WebRTC for IM (see Section 2.6.1). As WebRTC requires a browser, the open-source CefSharp (2019) library is used. CefSharp provides .NET bindings for the chromium embedded framework (CEF) via windows presentation foundation (WPF) or windows forms. In other words, it provides a chromium browser windows forms control.

TIA Portal uses windows forms. Thus, the chromium-browser is embedded via a control within a windows form. Using the chromium-browser increases the total memory usage. Memory measurements show that ∼180-220MB of additional RAM is needed, which accounts for ∼7% of the total RAM usage. EasyRTC is used as the WebRTC toolkit. Figure 25 illustrates an overview of the underlying architecture. Further, a clear separation of client and server parts is visible. On the server-side, we have the EasyRTC server module. This module already contains a signaling server and is written in Node.js. It is connected to a MySQL database, which saves user account information. On the client-side, we have the aforementioned integration of the CefSharp control within the TIA Portal. The CefSharp control accesses the chat via the server's IP address. The EasyRTC client-side chat consists of an HTML file and a JavaScript file. The HTML file defines actions, e.g., sending a chat message, and the GUI, the JavaScript file contains the program's logic. When the web-page is loaded, the corresponding JavaScript file is called to connect to the easyRTC server. Listing 3 depicts the JavaScript statements used upon connection establishment. Further, we can see two basic listeners used to indicate the online status of peers (RoomOccupantListener) and for receiving new messages (PeerListener). The connect method accepts the application name and a callback for a successful and unsuccessful connection. Displaying new messages within a conversation uses the document object model (DOM) to alter the underlying client-side HTML code dynamically.

```javascript
    easyrtc.setUsername(username);
    easyrtc.setPeerListener(receiveMessage);
    easyrtc.setRoomOccupantListener(showContacts);
    easyrtc.connect("tia-chat", connectSuccess, connectFailure);
```

Listing 3: Exemplary JavaScript code used to connect to easyRTC server

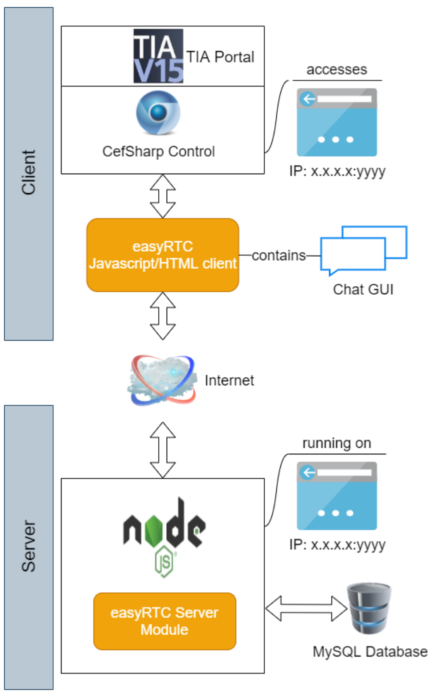


Figure 25: Approach 1 overview

TIA Portal users connect to that signaling server, as described above. The signaling server is then responsible for WebRTC connection establishment between the peers. After signaling has been carried out, the TIA Portal users communicate in a P2P manner. Within this approach, STUN/TURN is not considered. The signaling process is depicted in figure 26.

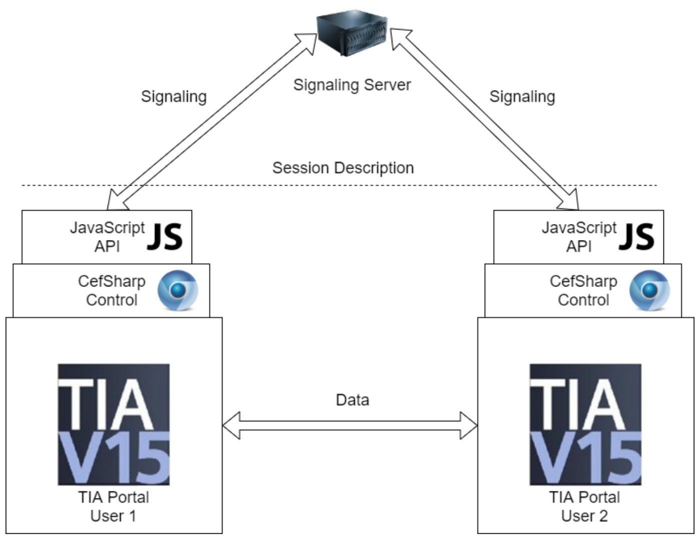
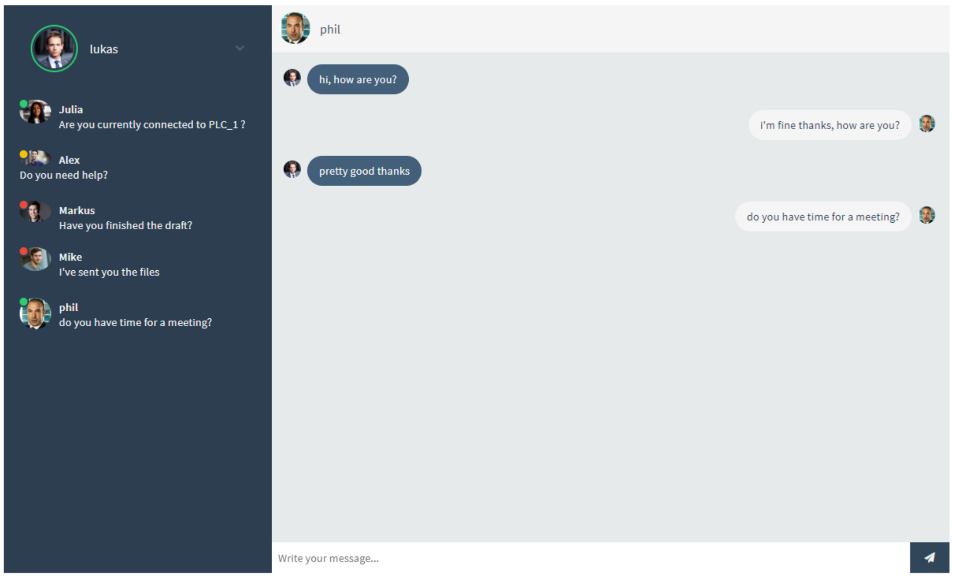


Figure 26: Approach 1 signaling

Figure 27 shows the GUI of the IMS. In the left pane, all currently connected users are shown. Each user has a name, a profile picture, and the text of the last message within the conversation. Each contact has a small colored icon showing his presence status. Currently, the presence status is not dynamically updated and just set hard-coded for demonstration purposes. Within the right pane, the chat history with a dedicated user is shown. The chat history is not saved. Group conversations are not supported.

The TIA Portal is endowed with a so-called multiuser feature, which allows up to 20 people to work collaboratively. In order to use this feature, clients need to set up a multiuser server. TIA Portal already provides extra tools that allow easy configuration and user management of the dedicated multiuser server. Instead of hosting an own server for IM, the currently described approach can be integrated into these tools. This provides the following benefits: no extra server is needed, reusable user management, and no additional costs. The integration into this multiuser feature constitutes a possible future enhancement.


lukas

Julia
Are you currently connected to PLC_1 ?

Alex
Do you need help?

Markus
Have you finished the draft?

Mike
I've sent you the files

phil
do you have time for a meeting?

phil

hi, how are you?

i'm fine thanks, how are you?

pretty good thanks

do you have time for a meeting?

Figure 27: Approach 1 GUI

7.2 Approach 2: Cloud-based solution with SignalR service

This approach aims to provide a cloud-based solution based on .NET SignalR. Azure was chosen as the designated cloud platform because it also provides an integrated SignalR service. The integration into the TIA Portal is implemented in the same way as in approach 1, which means with a CefSharp windows form control. This results in the same additional memory consumption, meaning ∼180-220MB of additional RAM or ∼7% of the total RAM usage. Figure 28 depicts the architecture overview of approach 2. We can see a clear separation between client and cloud parts. Azure hosts the whole ASP.NET Core web application, the SignalR service, and a database for saving user account information. The web application is implemented as a Microsoft ASP.NET Core web application, which uses Razor as a syntax view engine to create dynamic web pages. The web pages are implemented in files with a ".cshtml" suffix. Razor has its own syntax, but allows using HTML and JavaScript. The web application is available via a public URL structured in the Azure URL format, e.g., "https://tiachat.azurewebsites.net/". On the client-side, the CefSharp control accesses the web application via the URL above. Clients have a persistent connection to the SignalR service (see Section 2.6.2 for more information).

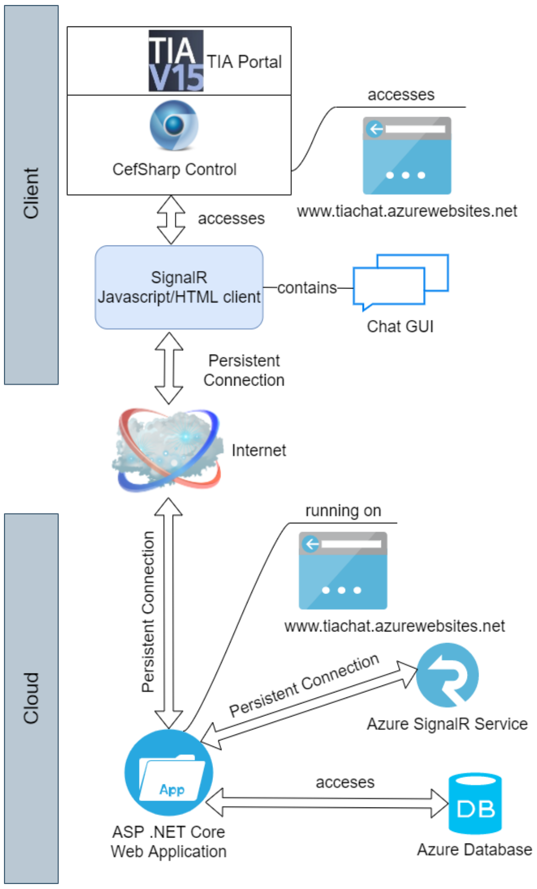


Figure 28: Approach 2 overview

The approach of hosting the whole application was chosen over using Azure functions because of the availability of tutorials and extended information. Azure was chosen as the target cloud because it provides easy integration of the SignalR service. Hosting the web application, together with the database and SignalR service, does not come for free. During implementation and testing phase, the free Azure pricing tier F1 (Microsoft Azure, 2020) was chosen. However, in production, this is not possible because only one hour per day of computation time is included. In the production phase, the chat service should be available during the whole day resulting in a more expensive price tier. We suggest using the B1 pricing tier, which includes 24 hours computation time per day, and costs €47 per month. A free price tier was chosen for the SignalR service, allowing up to 20 connected clients and 20000 messages per day. The cheapest SQL database price tier, costs €4,20 and contains one GB of storage. In total, this solution adds up to €51,2 per month. All Azure services are scalable, meaning that amounts of, e.g., client connections, can be easily adapted depending on the workload or traffic of the web application.

The underlying IMS is implemented as a simple chat room. Within figure 29, we can see multiple connected clients and how a broadcast messaged is transferred from one client to the others. Broadcast messages are used to notify already connected clients about new clients and to send additional messages to the chat room. Further, broadcast messages are relayed over the SignalR hub. The clients are connected via WebSockets. Listing 4 shows an example implementation of a SignalR Hub. Implementing an own Hub requires inheriting from the Microsoft.AspNetCore.SignalR.Hub base class. The hub contains a method used to broadcast messages to all connected clients. Listing 5 contains the corresponding code to access the hub within the web page. The signalR object is made accessible via the script tag in line 1. After the connection to the hub has been established, the onConnected method is called, which triggers a new broadcast message. We can see that the name of the hub's method needs to match the first parameter (case-insensitive) of the invoke method. The hub receives the broadcast message and forwards it all clients. Clients then need to process the incoming message.

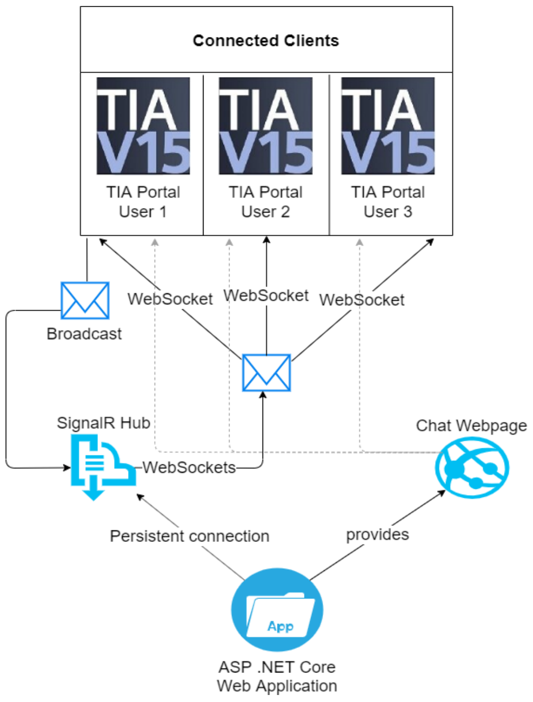


Figure 29: Approach 2 message transfer

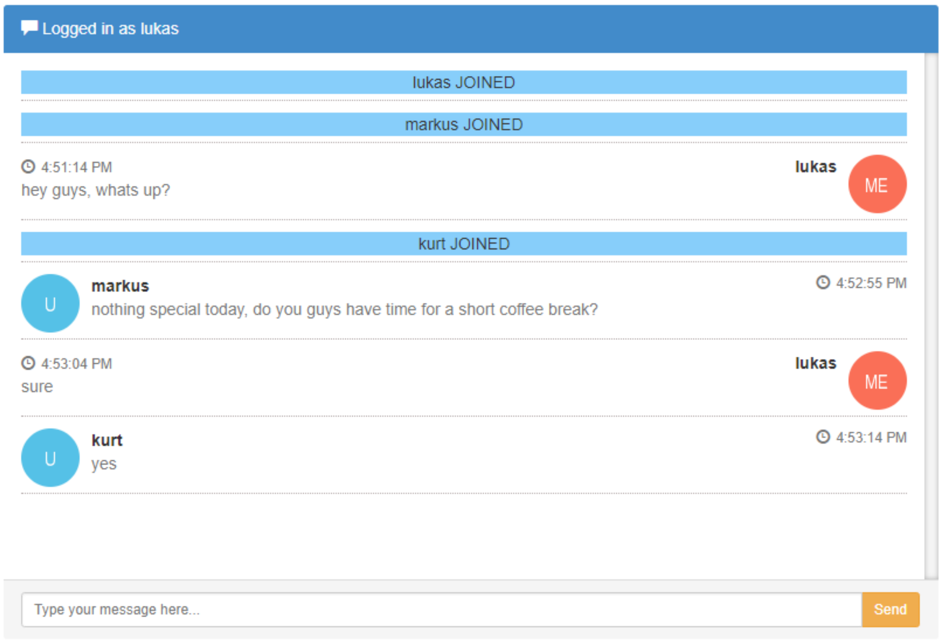
```
public class ChatHub : Hub
{
    public async Task BroadcastMessage(
        string name,
        string message)
    {
      await Clients.All.SendAsync("broadcastMessage", name, message);
    }
}
```

Listing 4: Exemplary SignalR Hub code

```
<script type="text/javascript" src="https://../signalr.min.js"></script>

var connection = new signalR.HubConnectionBuilder()
    .withUrl('/chat')
    .build();

function onConnected(connection) {
    connection.invoke('broadcastMessage', '_SYSTEM_', username);
    }

connection.start()
    .then(function() {
        onConnected(connection);
    })
```

Listing 5: SignalR connection establishment

Figure 30 shows the GUI of the IMS. As already mentioned, the GUI looks similar to a chatroom. Within the conversation history, we see notifications about users that joined. Based on the concept of a chatroom, all users write their messages to the same conversation. Further, each message has a timestamp, username, and an icon indicating whether the current user wrote the message or it was sent by another user. Due to time constraints, no further features were implemented.


Logged in as lukas

lukas JOINED

markus JOINED

4:51:14 PM
hey guys, whats up?
lukas ME

kurt JOINED

U markus
nothing special today, do you guys have time for a short coffee break?
4:52:55 PM

4:53:04 PM
sure
lukas ME

U kurt
yes
4:53:14 PM

Figure 30: Approach 2 GUI

7.3 Approach 3: Chat via PLC

The idea of approach three is to use connected PLCs for IM rather than using any 3rd party components, such as a browser control or WebRTC libraries. Therefore, chat messages should be directly transferred via a PLC. Within the TIA Portal there exists a framework that makes certain information of the underlying automation system available to the outside where it is accessed by clients. Let us refer to it as the online object model (OOM). It is used in various application areas, such as embedded applications of CPUs, HMIs, panels as well as in PC based applications of the engineering system. OOM allows us to establish and manage connections between clients and servers on application level, download, upload, access and store objects like Blocks, system status, diagnosis buffer, and notify sets of attributes and objects cyclically or upon change like alarms or system events. Clients use OOM's API to access information. This approach uses OOM to write chat messages to a specific attribute of an online object. For this purpose, two service of OOM are used: the object access service and attribute service. The object access service is used to obtain the dedicated object, where the chat messages are written to a specific attribute. The attribute service allows us to get/set variables on an object and to get notified on attribute changes. The PLC's online object contains attributes such as the name, unique id or firmware version. This approach reuses an existing example attribute. Various flags are defined on attributes, such as a persistence flag indicating whether the attribute is saved on a memory card. Any attribute having the persistence flag needs an extra download to the PLC to apply changes. Applying a download after each new message introduces too much overhead. Therefore, the used example attribute is not flagged with the persistence specifier, which means that the contents are discarded when the PLC is switched off. The content of the attribute is saved in plaintext. Conversation history is currently limited by two megabytes allowing a total amount of 1048576 characters because a character accounts for two bytes in Unicode. Two megabytes was chosen because the PLC's internal memory is limited and not to jeopardize PLC cycle times.

Figure 31 shows an overview of approach three. Within the TIA Portal a new windows form control has been created. This control is used to show the chat conversation for each PLC. It can be opened via the shortcut: CTRL+3. Within the chat control, only the chat history of currently connected PLCs is accessible. Further, we see that only the OOM of connected PLCs is accessible. This means that messaging via PLC_3 is only possible after a connection has been established. Messaging via PLC_1 and PLC_2 is possible.

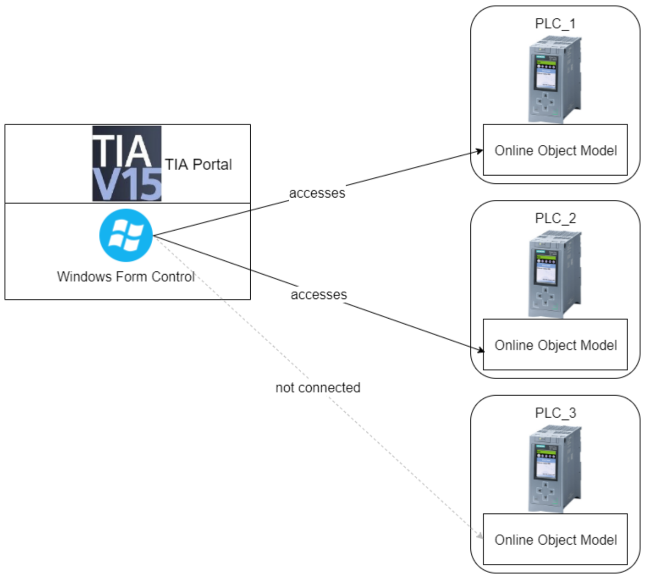


Figure 31: Approach 3 overview

Figure 32 illustrates the general workflow of this approach with two TIA Portal users. In steps 1 and 4, the users connect to a dedicated PLC. We can see that a publish/subscribe mechanism is used. Within steps 3 and 6, the TIA Portal users subscribe for attribute changes. In step 7, TIA Portal user 1 writes a new message and, afterwards all connected users are notified about the new message. This workflow bears one small problem. When a TIA Portal user compiles and downloads the program on a PLC, other users lose their connection to the PLC. While the whole process can take several minutes, the users want to connect again to the PLC to participate in the chat. However, this is only possible when the download has finished. One open problem of the TIA Portal is that users do not get notified when re-connection is possible.

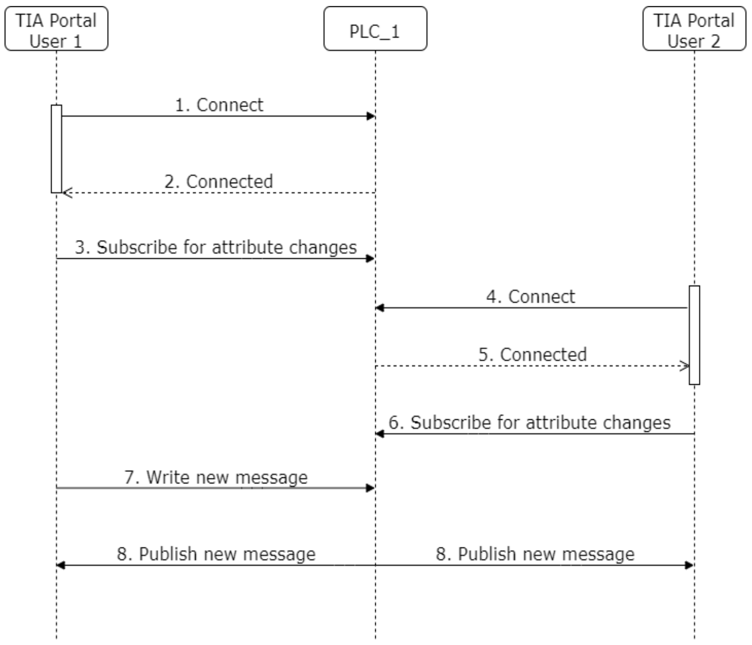
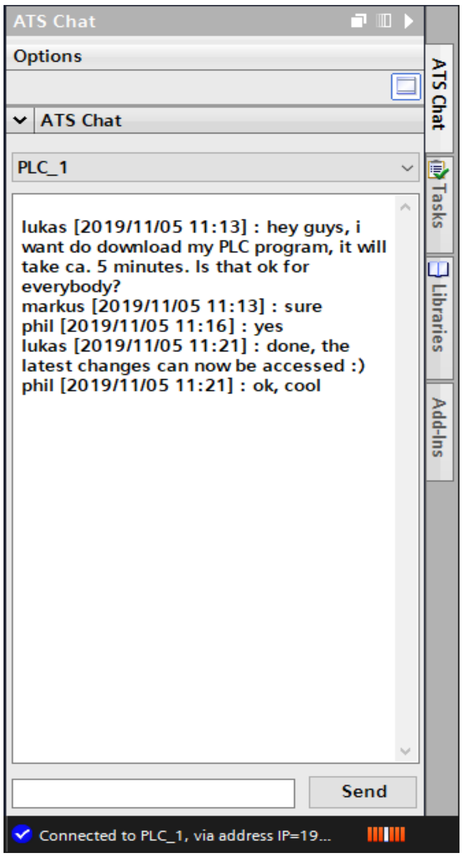


Figure 32: Approach 3 workflow

The GUI of this approach is depicted in figure 33. As already described, it is implemented as a windows form and contains a drop-down list for PLC selection and a text box for displaying chat messages. It is implemented as a so-called *taskcard*. The taskcard concept refers to views that are available in the right pane within every TIA Portal view. The GUI aims to provide a basic design that conforms to the corporate look and feel of TIA Portal. We can see that each message contains a username and a timestamp. The username refers to the TIA Portal username that can be customized within the settings of the TIA Portal. It is not possible to apply any formatting within the windows text box. Another limitation that arises from the TIA Portal is that it is not possible to obtain all currently connected users of a PLC. A simple notification mechanism has been implemented that changes the background color of the right pane if the chat is currently not opened.



lukas [2019/11/05 11:13] : hey guys, i want do download my PLC program, it will take ca. 5 minutes. Is that ok for everybody?
markus [2019/11/05 11:13] : sure
phil [2019/11/05 11:16] : yes
lukas [2019/11/05 11:21] : done, the latest changes can now be accessed :)
phil [2019/11/05 11:21] : ok, cool

Connected to PLC_1, via address IP=19...

Figure 33: Approach 3 GUI

8 Evaluation

This chapter evaluates the three different approaches that have been implemented within the scope of this thesis. The result is shown in a feature matrix in table 3 in Section 8.4. Eighteen features are used to evaluate the approaches. This section provides a textual description of the different features together with an explanation of why the corresponding feature is used in the evaluation:

1. Availability: Percentage of a specified time interval, where the system was available for normal use. In general, the IMS should provide high availability. Based on requirement 15.
   - low: 0-50%
   - medium: 50-90%
   - high: 90-99.9%, e.g., 99.9% is guaranteed by Azure's service level agreement
2. Scalability: Maximal number of concurrent supported users. Within this scenario, the IMS needs to support up to 20 concurrent users, which might be considered as a very low number, but within this context, it is not necessary to support a higher number of users. Based on requirement 17.
3. Extensibility: Defines the IMS's ability to be extended with new functionality and the corresponding level of effort required to implemented the new feature. Based on requirement 12.
   - low: The system is very limited and can only be extended really hard.
   - medium: New features can be implemented with mediocre effort.
   - high: A variety of new features can easily be implemented.
4. Maintenance: Describes the components that need to be maintained by the customer. The implemented IMS should be maintainable with low effort. Based on requirement 16.
5. Additional memory requirements: Describes the amount and type of memory needed for the IMS. A requirement for memory usage has not been defined, but low memory usage is a common goal.
6. Cost/Month in Euro: Describes the involved costs needed for using the IMS. No additional costs should arise for Siemens. Based on requirement 10.
7. Use of third-party components: Describes whether the IMS uses third-party components. No dedicated requirement is defined. Third-party components are allowed if they are open-source, but using such components results in more maintenance effort for developers.
8. Message confidentiality: Describes whether the messages are sent encrypted. Based on requirement 13.

9. Authenticity: Defines the ability to identify a user or to verify his claimed identity.
10. Privacy: Is a branch of data security focusing on proper data handling. Three privacy properties regarding IMS defined by Patil and Kobsa (2004) are used. Based on requirement 11.
    - Non-Contact: Non-contacts are not able to contact a user.
    - Availability: Presence status indicator to prevent frequent interruptions when the user is busy.
    - Content: Indicates whether messages are saved in plaintext, e.g., in a database.
11. Setup difficulty / Installability: Describes the effort needed to set up the IMS. The implemented IMS aims for a zero-config approach. Too complicated installation or configuration could lead to abandonment. Based on requirement 14.
    - low: Installation is not required.
    - medium: Less than 10 installation steps are needed.
    - high: More than 10 installation steps are needed.
12. Additional NAT handling needed: Indicates whether additional NAT handling is needed in order to ensure correct execution in different NAT scenarios. As already mentioned, the IMS should not need complex configuration. Based on requirement 14.
13. One-on-one messaging: Functionality to chat with each user individually. This is considered as a standard IM feature. Based on requirement 1.
14. Presence list: The presence status of other users should be visible. Based on requirement 2.
15. History persistence: Describes the IMS's capability of persisting history such that conversation history is available for users after re-connection. The absence of history persistence results in loss of whole conversations. Based on requirement 3.
16. Chat room: Functionality to chat in a group. This feature improves collaboration because multiple automation engineers can participate in a conversation. Based on requirement 4.
17. Notifications: Users should get notified of new messages. Without notifications, messages can simply be overseen, potentially leading to delays and a decrease in productivity. Based on requirement 5.
18. Technology: Describes the different technologies used within the approaches. Only technologies that allow the integration of the IMS in the TIA Portal are allowed. Based on requirement 9.

The functional requirements with a low priority (6-8) have not been included in the implementation due to time constraints. Hence, they are also not included in the evaluation.

	Approach 1	Approach 2	Approach 3
Availability	medium	high	medium
Scalability (#Users)	max. 20	max. 20	max. 20
Extensibility	high	high	low
Maintenance	Server	Azure Services	PLC
Additional memory requirements	~180-220MB RAM	~180-220MB RAM	2MB CPU internal memory
Cost/Month in Euro	free	51,2	free
Use of third party components	yes	yes	no
Message confidentiality	yes	yes	no
Authenticity	yes	yes	no
Privacy	Non-Contact, Content	Availability	Availability
Setup difficulty / Installability	medium	low	low
Additional NAT handling needed	yes	no	no
One-on-one messaging	yes	not implemented	no
Presence List	yes	not implemented	not implemented
Chat Room	not implemented	yes	yes
History Persistence	not implemented	not implemented	yes
Notifications	not implemented	not implemented	yes
Technology	WebRTC + Node.js + CefSharp	Azure Cloud with SignalR + CefSharp	PLC properties + TIA .Net Control

Table 3: Feature Matrix

8.1 Discussion of Approach 1

Approach 1 fulfills the requirements with the highest priority: 1, 9, 10, 11, 12, and 13. The features chat room, history persistence, and notifications have not been implemented due to time constraints. Based on the high extensibility of this approach, such new features can easily be implemented. As this approach is tightly coupled with the TIA portal multiuser feature, it is not available to customers without this feature. However, the standard use case for automation engineers working collaboratively on the same PLCs, HMIs, etc., is via the multiuser feature. Customers need to maintain their own multiuser server hosting the multiuser software, making it an ideal target for integrating an IMS. This means that no additional server needs to be maintained by Siemens or a customer. Currently, the multiuser feature supports up to 20 users. As the signaling process runs on the same server, the approach is limited by the amount of memory and number of accessible ports. Also, sending/receiving times depend on the server's network connection. The evaluation showed that this approach is the only one needing additional NAT handling. This approach requires a future STUN/TURN concept to be applicable in complex NAT scenarios.

8.2 Discussion of Approach 2

Approach 2 fulfills the requirements with the highest priority: 1, 9, 11, 12, and 13. High scalability and availability, together with low maintenance effort, is provided through the use of the Azure cloud. However, in this scenario, large amounts of users are not expected. The high costs resulting from the Azure services contradict with requirement 10 and thus, make this approach impracticable.

8.3 Discussion of Approach 3

Approach 3 fulfills the requirements with the highest priority: 1, 9, 10, and 11. This approach is tightly coupled to PLC properties. Hence limitations, based on the capabilities of a PLC, arise. This approach provides the benefit that no server or cloud service is needed, resulting in no additional costs and low maintenance effort. In order to not to jeopardize PLC cycle times, messages are not encrypted contradicting with requirement 13. This approach provides with history persistence (up to 2MB), a chat room and notifications a broad feature scope, but new features, such as file transfer, cannot easily be extended, contradicting with requirement 12. Another problem arises due to the PLC download workflow. When a TIA portal user compiles and downloads the program on a PLC, other users lose their connection to the PLC, resulting in the unavailability of the IMS.

8.4 Result

The evaluation has shown that the primary choice is approach 1, as the requirements with the highest priority are fulfilled. Although several functional requirements with a lower priority have not been implemented, they can easily be implemented due to the high extensibility of this approach. The main counter-argument for approach 2 is the involved costs making this approach impracticable. Approach 3 is the secondary choice. Due to the restrictions that arise from the PLCs, messages are not send encrypted and as already mentioned, a presence list and one-on-one messaging is not possible based on TIA Portal limitations. Further, the workflow problem results in unavailability of the IMS. However, it is possible to integrate both approaches into the TIA Portal. Approach 3 would serve as a basic IMS for all TIA Portal users and approach 1 as an extended IMS targeting only customers using the TIA Portal Multiuser feature.

9 Limitations

The approaches implemented within the scope of this thesis have been explained in Chapter 7 and evaluated in Chapter 8. This chapter explains the limitations of the different approaches. In general, the goal was not the implementation of a whole new IMS with a broad feature-scope, but rather to evaluate state-of-the-art technologies and how they can be used to integrate a basic IMS into the TIA Portal.

9.1 Compatibility/Technology/Dependencies

Protocols and components used within the different approaches have been chosen to fulfill requirement 9 - TIA Portal integration. As the TIA Portal is based on the .Net framework, all presented approaches provide .NET compatibility. In the following used technologies and related dependencies of the different approaches are explained.

- Approach 1: The opensource framework EasyRTC is used to enable real-time communication via WebRTC. Signaling server is implemented in Node.js. A CefSharp .Net browser control is integrated into the TIA Portal to access the website, where the IM service runs. This approach is tightly coupled with the TIA Portal Multiuser feature. Hence, access to the IMS is limited to customers using this feature.
- Approach 2: This approach heavily relies on Azure and its SignalR service. The same CefSharp browser control as in approach 1 is used for TIA Portal integration and website access.
- Approach 3: The dependency of this approach arises through the use of Siemens Simatic PLCs for IM. In terms of availability, users need to be connected to a dedicated PLC, which is not possible in certain scenarios. GUI integration is implemented via a TIA .NET control.

9.2 Scalability

As seen in Chapter 8, all three approaches do not provide scalability of up to several thousands users. All approaches support currently up 20 users, which might be considered as a very low number, but within this context, it is not necessary to support a higher number of users. Approach 1 builds upon the TIA Portal Multiuser feature, which currently supports up to 20 users. Hence, the number of supported users for IM goes hand-in-hand with this limitation. Increasing this number within the multiuser feature also increases the amount supported IM users. The limitation of 20 users in approach 2 arises, due to the use of Azure's SignalR service with a free pricing tier. Switching to a higher pricing tier increases the number of supported users, but also increases related costs. Scalability within the second approach can be easily increased with more money. This limitation only comes from the SignalR supported Users. In approach 3, the number of supported users is limited to 20 to no to jeopardize PLC cycle times. Customers using more than a single PLC, can use each PLC for IM, allowing to increase the total number of supported users.

9.3 Usability

Based on the requirement that the IMS needs to be integrated within the TIA Portal, usability shortcomings arise. In general, a low degree of freedom in GUI design exists, because it must comply with the TIA Portal's corporate look and feel. One usability issue emerges through the limited number of available shortcuts. Intuitive shortcuts are already reserved for other features, allowing using only less-intuitive shortcuts, e.g., CTRL+ALT+9 for opening the IMS.

9.4 Functionality

All approaches are implemented based on the proof-of-concept paradigm, demonstrating the feasibility of the individual approaches. This includes the absence of implemented unit/integration/UI tests. Due to time constraints several nice-to-have IM features, that might be expected from a state-of-the-art IMS, e.g., file transfer or multiple group chats are not implemented.

9.5 Process/Maintenance

Opensource software used within the TIA Portal needs to run through a license and security clearing. Only successfully cleared libraries are allowed to be included. Currently, new TIA Portal versions come out every year. In a new version, use of opensource software needs to be re-evaluated and potentially updated to a new version. This means that if problems are encountered with a specific library, customers need to wait up to one year to get a new version, depending on the problem's severity.

10 Conclusion and Future Work

Over the past decades, various IMS and their enabling technologies have been implemented. This thesis focused on how to use those technologies to design and integrate an IMS into the enterprise automation software TIA Portal. The positive and negative work-related effects of IMS are pointed out within this thesis. In the TIA Portal's context, an integrated IMS improves the automation process by providing a unified communication process and, thus, decreases communication overhead arising through the use of various CMC types. Within this thesis, three different approaches have been established. Ideas, used technologies, architectures, protocols, and GUI designs of the individual approaches are illustrated. The different approaches have been evaluated via a feature matrix. The evaluation showed that approach 1 and 3 are the primary choices according to the defined requirements. Approach 2 is not applicable due to related costs.

The main goal for the future is to integrate one of the approaches into the TIA portal. All approaches have been implemented in a prototype-fashion. In order to reach production-ready product quality, the following base actions need to be taken: conduct code reviews, implement unit/integration/UI tests, execute usability tests. For approach 1, the integration into the TIA Portal Multiuser feature needs to be implemented. Secondly, a STUN/TURN server concept needs to be developed to enable IM in complex NAT setups. Potential future work for approach 3 is the retrieval and display of all currently connected users of a PLC within the IMS. The retrieval of connected users is not yet implemented within the TIA Portal. Further, a new OOM attribute needs to be introduced on PLCs, because an example attribute was used, which cannot be used in production. Finally, to provide a generic solution for Siemens Simatic PLCs, performance tests with different PLCs need to be taken to evaluate the IM impact.

Acronyms

AWS Amazon Web Services

BaaS Backend As A Service

CEF Chromium Embedded Framework

CMC Computer-mediated Communication

DDoS Distributed Denial Of Service

DHT Distributed Hash Table

DOM Document Object Model

FaaS Function As A Service

GUI Graphical User Interface

HMI Human-machine-interface

IaaS Infrastructure As A Service

IM Instant Messaging

IMS Instant Messaging System

JID JabberID

MES Manufacturing Execution System

MPI Multi Point Interface

NAT Network Address Translation

OOM Online Object Model

OpenPGP Open Pretty Good Privacy

OS Operating System

OTR Off-the-Record

P2P Peer-to-peer

PaaS Platform As A Service

PLC Programmable Logic Controller

Profibus Process Field Bus

REST Representational State Transfer

RTC Real-time Communication

SaaS Software As A Service

SASL Simple Authentication And Security Layer

SDP Session Description Protocol

SIL Safety Integrity Level

SIMPLE SIP For IM And Presence Leverage Extension

SIP Session Initiation Protocol

S/MIME Secure/Multipurpose Internet Mail Extensions

SOAP Simple Object Access Protocol

STUN Session Traversal Utilities For NAT

TIA Portal Totally Integrated Automation Portal

TLS Transport Layer Security

TURN Traversal Using Relays Around NAT

UA User Agent

UI User Interface

URI Uniform Resource Identifier

VM Virtual Machine

VMM Virtual Machine Monitor

WebRTC Web Real-time Communication

WPF Windows Presentation Foundation

WSDL Web Services Description Language

XMPP Extensible Messaging And Presence Protocol

References

Abu-Salma, R., Sasse, M. A., Bonneau, J., & Smith, M. (2015). POSTER: Secure Chat for the Masses? User-centered Security to the Rescue. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (pp. 1623–1625). ACM. doi:10.1145/2810103.2810126

Adzic, G. & Chatley, R. (2017). Serverless computing: Economic and architectural impact. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering* (pp. 884–889). ACM. doi:10.1145/2810103.2810126

Aguilar, J. M. (2014). *SignalR programming in microsoft ASP.NET*. Microsoft Press. ISBN: 978-0735683884.

Åkerman, M. (2018). *Implementing shop floor IT for industry 4.0* (Doctoral dissertation, Chalmers University of Technology). Retrieved December 1, 2019, from https://www.researchgate.net/publication/326224890_Implementing_Shop_Floor_IT_for_Industry_40

Amazon. (2014). AWS lambda. Retrieved June 10, 2019, from https://aws.amazon.com/lambda/

Anderson, C. (2015). Docker [Software engineering]. *IEEE Software*, *32*(3), 102–c3. doi:10.1109/MS.2015.62

Androutsellis-Theotokis, S. & Spinellis, D. (2004). A survey of peer-to-peer content distribution technologies. *ACM computing surveys (CSUR)*, *36*(4), 335–371. doi:10.1145/1041680.1041681

ANSI/ISA-95.00.03-2005. (2005). *Enterprise control system integration part 3: Activity models of manufacturing operations management*. Tech. Rep. Netherlands.

Armoogum, S. & Mudhoo, S. K. (2016). A Secure Messaging and File Transfer Application. *ICCGI 2016, The Eleventh International Multi-Conference on Computing in the Global Information Technology*, 42–47.

Bakar, H. S. A. & Johari, N. A. H. (2009). Instant Messaging: The Next Best Knowledge Sharing Tools in a Workplace After E-mail? In *Proc. 2nd IEEE Int. Conf. Computer Science and Information Technology* (pp. 268–269). doi:10.1109/ICCSIT.2009.5234577

Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., . . . Slominski, A. (2017). Serverless Computing: Current Trends and Open Problems. In *Research Advances in Cloud Computing* (pp. 1–20). Springer. doi:10.1007/978-981-10-5026-8_1

Barry, B. I. A. & Tom, F. M. (2011). Instant Messaging: Standards, Protocols, Applications, and Research Directions. In B. Kutais (Ed.), *Internet Policies & Issues* (Chap. 8). Nova Science Publishers Inc. ISBN: 1616687452.

Bell, R. (2006). Introduction to IEC 61508. In *Proceedings of the 10th australian workshop on safety critical systems and software-volume 55* (pp. 3–12). Australian Computer Society, Inc. ISBN: 1-920-68237-6.

Bernstein, D. (2014). Containers and cloud: From LXC to docker to kubernetes. *IEEE Cloud Computing*, *1*(3), 81–84. doi:10.1109/MCC.2014.51

Bilal, K., Khalid, O., Erbad, A., & Khan, S. U. (2018). Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers. *Computer Networks*, *130*, 94–120. doi:10.1016/j.comnet.2017.10.002

Bischofs, L., Hasselbring, W., & Warns, T. (2008). Peer-to-Peer-Architekturen. In *Handbuch der Software Architekturen*. dPunkt Verlag. ISBN: 978-3898643726.

Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. (2012). Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on mobile cloud computing* (pp. 13–16). ACM. doi:10.1145/2342509.2342513

Borisov, N., Goldberg, I., & Brewer, E. (2004). Off-the-record communication, or, why not to use PGP. In *Proceedings of the 2004 acm workshop on privacy in the electronic society* (pp. 77–84). ACM. doi:10.1145/1029179.1029200

Buxmann, P., Hess, T., & Lehmann, S. (2008). Software as a service. *Wirtschaftsinformatik*, *50*(6), 500–503. doi:10.1007/s11576-008-0095-0

Callas, J., Donnerhacke, L., Finney, H., & Thayer, R. (1998). RFC 2440: OpenPGP message format. *Internet Engineering Task Force*. Retrieved from https://tools.ietf.org/html/rfc2440

Camarillo, G. [Gonzalo]. (2009). RFC 5694: Peer-to-peer (P2P) architecture: Definition, taxonomies, examples, and applicability. *Network Working Group*. Retrieved from https://tools.ietf.org/html/rfc5694

Campbell, B., Rosenberg, J., Schulzrinne, H., Huitema, C., & Gurle, D. (2002). RFC 3428: Session initiation protocol (sip) extension for instant messaging. *Internet Engineering Task Force*. Retrieved from https://tools.ietf.org/html/rfc3428

CefSharp. (2019). CefSharp. Retrieved November 6, 2019, from https://github.com/cefsharp/CefSharp

Chang, W. Y., Abu-Amara, H., & Sanford, J. F. (2010). *Transforming enterprise cloud services*. Springer Netherlands. ISBN: 9789048198467.

Chinnici, R., Moreau, J.-J., Ryman, A., & Weerawarana, S. (2007). Web services description language (WSDL) version 2.0 part 1: Core language. *W3C recommendation*, *26*(1), 19.

Combe, T., Martin, A., & Di Pietro, R. (2016). To Docker or not to Docker: A Security Perspective. *IEEE Cloud Computing*, *3*(5), 54–62. doi:10.1109/MCC.2016.100

Cooper, A., Reimann, R., & Cronin, D. (2007). *About face 3: The essentials of interaction design*. John Wiley & Sons. ISBN: 978-0470084113.

Cramer, C., Kutzner, K., & Fuhrmann, T. (2004). Bootstrapping locality-aware p2p networks. In *Proceedings. 2004 12th IEEE international conference on networks (ICON 2004)* (Vol. 1, pp. 357–361). IEEE. doi:10.1109/ICON.2004.1409169

Cumming, J. (2003). SIP market overview. Tech. rep. Retrieved November 30, 2019, from http://docshare01.docshare.tips/files/25252/252522914.pdf

Curry, S. J. J. (2013). Instant-messaging security. In J. R. Vacca (Ed.), *Computer and information security handbook* (Third Edition, pp. 727–740). Boston: Morgan Kaufmann. ISBN: 978-0-12-803843-7. doi:10.1016/B978-0-12-803843-7.00051-X

Czerwinski, M., Cutrell, E., & Horvitz, E. (2000a). Instant messaging and interruption: Influence of task type on performance. OZCHI 2000 Conference Proceedings. Association for Computing Machinery, Inc. Retrieved from https://www.microsoft.com/en-us/research/publication/instant-messaging-and-interruption-influence-of-task-type-on-performance/

Czerwinski, M., Cutrell, E., & Horvitz, E. (2000b). Instant messaging: Effects of relevance and timing. In *People and computers xiv: Proceedings of hci 2000* (People and Computers XIV: Proceedings of HCI 2000, Vol. 2, pp. 71–76). Retrieved from https://www.microsoft.com/en-us/research/publication/instant-messaging-effects-of-relevance-and-timing/

De Luca, A., Das, S., Ortlieb, M., Ion, I., & Laurie, B. (2016). Expert and non-expert attitudes towards (secure) instant messaging. In *Twelfth symposium on usable privacy and security (SOUPS 2016)* (pp. 147–157). ISBN: 978-1-931971-31-7.

Debbabi, M. & Rahman, M. (2003). The war of presence and instant messaging: Right protocols and APIs. In *2003 international symposium on VLSI technology, systems and applications. proceedings of technical papers.* IEEE. doi:10.1109/ccnc.2004.1286884

Dinger, J. & Waldhorst, O. P. (2009). Decentralized bootstrapping of P2P systems: A practical view. In *International conference on research in networking* (pp. 703–715). Springer. doi:10.1007/978-3-642-01399-7_55

Dittrich, Y. & Giuffrida, R. (2011). Exploring the role of instant messaging in a global software development project. In *2011 IEEE sixth international conference on global software engineering* (pp. 103–112). doi:10.1109/ICGSE.2011.21

Dolui, K. & Datta, S. K. (2017). Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In *2017 global internet of things summit (GIoTS)* (pp. 1–6). IEEE. doi:10.1109/GIOTS.2017.8016213

Dua, R., Raja, A. R., & Kakadia, D. (2014). Virtualization vs containerization to support PaaS. In *2014 IEEE international conference on cloud engineering* (pp. 610–614). IEEE. doi:10.1109/IC2E.2014.41

Eberspächer, J. & Schollmeier, R. (2005). First and second generation of peer-to-peer systems. In *Peer-to-peer systems and applications* (pp. 35–56). Springer. doi:10.1007/11530657_5

Ermoshina, K., Musiani, F., & Halpin, H. (2016). End-to-end encrypted messaging protocols: An overview. In *International conference on internet science* (pp. 244–254). Springer. doi:10.1007/978-3-319-45982-0_22

Fette, I. & Melnikov, A. (2011). RFC 6455: The websocket protocol. *IETF*. Retrieved from https://tools.ietf.org/html/rfc6455

Fox, G. C., Ishakian, V., Muthusamy, V., & Slominski, A. (2017). Status of serverless computing and function-as-a-service(faas) in industry and research. *ICDCS 2017 Workshop*. doi:10.13140/rg.2.2.15007.87206

Furht, B. (2010). Cloud computing fundamentals. In *Handbook of cloud computing* (pp. 3–19). Springer. doi:10.1007/978-1-4419-6524-0

Fussell, S. R., Kiesler, S., Setlock, L. D., & Scupelli, P. (2004). Effects of instant messaging on the management of multiple project trajectories. In *Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 191–198). ACM. doi:10.1145/985692.985717

Galitz, W. O. (2007). *The essential guide to user interface design: An introduction to gui design principles and techniques*. John Wiley & Sons. ISBN: 0470053429.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0-201-63361-2.

Garrett, R. K. & Danziger, J. N. (2007). IM = interruption management? Instant messaging and disruption in the workplace. *Journal of Computer-Mediated Communication*, *13*(1), 23–42. doi:10.1111/j.1083-6101.2007.00384.x

Geneiatakis, D., Kambourakis, G., Dagiuklas, T., Lambrinoudakis, C., & Gritzalis, S. S. (2005). SIP security mechanisms : A state-of-the-art review. In *Proceedings of the fifth international network conference 2005 (inc 2005)*. doi:10.1109/ICTTA.2006.1684926

Gnutella Developer Forum. (2003). Gnutella: A protocol for a revolution. http://rfc-gnutella.sourceforge.net/.

Gupta, A., Li, H., & Sharda, R. (2013). Should I send this message? understanding the impact of interruptions, social hierarchy and perceived task complexity on user performance and perceived workload. *Decision Support Systems*, *55*(1), 135–145. doi:10.1016/j.dss.2012.12.035

Hauswirth, M. & Dustdar, S. (2005). Peer-to-peer: Grundlagen und Architektur. *Datenbank-Spektrum*, *13*(2005), 5–13.

Hendrickson, S., Sturdevant, S., Harter, T., Venkataramani, V., Arpaci-Dusseau, A. C., & Arpaci-Dusseau, R. H. (2016). Serverless computation with openlambda. In *Proceedings of the 8th usenix conference on hot topics in cloud computing* (pp. 33–39). HotCloud'16. Denver, CO: USENIX Association. Retrieved from http://dl.acm.org/citation.cfm?id=3027041.3027047

Hönlinger, J. (2018). *The role of instant messenger as computermediated communication tool for knowledge sharing and teamwork performance* (Master's thesis, Joenkoeping University).

Isaacs, E., Walendowski, A., Whittaker, S., Schiano, D. J., & Kamm, C. (2002). The character, functions, and styles of instant messaging in the workplace. In *Proceedings of the 2002 ACM conference on computer supported cooperative work* (pp. 11–20). ACM. doi:10.1145/587078.587081

Jaanu, T., Paasivaara, M., & Lassenius, C. (2012). Near-synchronicity and distance: Instant messaging as a medium for global software engineering. In *2012 IEEE seventh international conference on global software engineering* (pp. 149–153). doi:10.1109/ICGSE.2012.37

Jadeja, Y. & Modi, K. (2012). Cloud computing-concepts, architecture and challenges. In *2012 international conference on computing, electronics and electrical technologies (ICCEET)* (pp. 877–880). IEEE. doi:10.1109/ICCEET.2012.6203873

Jennings, R. B., Nahum, E. M., Olshefski, D. P., Saha, D., Shae, Z.-Y., & Waters, C. (2006). A study of internet instant messaging and chat protocols. *IEEE Network*, *20*(4), 16–21. doi:10.1109/MNET.2006.1668399

John, K.-H. & Tiegelkamp, M. (2009). Die Programmiersprachen der IEC 61131-3. In *SPS-Programmierung mit IEC 61131-3* (pp. 103–211). Springer. doi:10.1007/978-3-642-00269-4_4

Josuttis, N. M. (2007). *SOA in practice: The art of distributed system design.* O'Reilly Media, Inc. ISBN: 0596529554.

Kamel, M., Scoglio, C., & Easton, T. (2007). Optimal topology design for overlay networks. In *Networking 2007. ad hoc and sensor networks, wireless* *networks, next generation internet* (pp. 714–725). Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-72606-7.

Lane, K. (2015). Overview of the backend as a service (BaaS) space. *API Evangelist*. Retrieved November 30, 2019, from https://apievangelist.com/2013/05/03/overview-of-the-backend-as-a-service-baas-space/

Lebbon, A. R. & Sigurjónsson, J. G. (2016). Debunking the instant messaging myth? *International Journal of Information Management*, *36*(3), 433–440. doi:https://doi.org/10.1016/j.ijinfomgt.2016.02.003

Levin, O. & Camarillo, G. (2006). RFC 4574: The session description protocol (SDP) label attribute. *Network Working Group*. Retrieved from https://tools.ietf.org/html/rfc4574

Li, H., Gupta, A., Luo, X., & Warkentin, M. (2011). Exploring the impact of instant messaging on subjective task complexity and user satisfaction. *European Journal of Information Systems*, *20*(2), 139–155. doi:10.1057/ejis.2010.59

Liu, Q. & Sun, X. (2012). Research of web real-time communication based on web socket. *International Journal of Communications, Network and System Sciences*, *5*(12), 797. doi:10.4236/ijcns.2012.512083

Loreto, S., Saint-Andre, P., Salsano, S., & Wilkins, G. (2011). RFC 6202: Known issues and best practices for the use of long polling and streaming in bidirectional http. *Internet Engineering Task Force*, *6202*(2070-1721), 32. Retrieved from https://tools.ietf.org/html/rfc6202

Mannan, M. & van Oorschot, P. C. (2004). Secure public instant messaging: A survey. *Proceedings of Privacy, Security and Trust*, 95.

Mansi, G. & Levy, Y. (2013). Do instant messaging interruptions help or hinder knowledge workers' task performance? *International Journal of Information Management*, *33*(3), 591–596. doi:10.1016/j.ijinfomgt.2013.01.011

Meletiadou, A. (2010). *Moderne Instant-Messaging-Systeme als Plattform für sicherheitskritische kollaborative Anwendungen* (doctoralthesis, University Coblenz).

Mell, P. & Grance, T. (2011). The NIST definition of cloud computing. *Communications of the ACM*, *53*. doi:10.6028/NIST.SP.800-145

Melnikov, A. & Zeilenga, K. (2006). RFC 4422: Simple authentication and security layer (SASL). *Internet Engineering Task Force*, *15*, 20. Retrieved from https://tools.ietf.org/html/rfc4422

Menascé, D. A. (2005). Virtualization: Concepts, applications, and performance modeling. In *Int. CMG conference* (pp. 407–414).

Microsoft Azure. (2020). Azure pricing. Retrieved January 11, 2020, from https://azure.microsoft.com/en-us/pricing/details/app-service/windows/

Microsoft Docs. (2018). Cloud service models. Retrieved November 9, 2019, from https://docs.microsoft.com/en-us/learn/modules/align-requirements-in-azure/3-service-models

Musiani, F. & Ermoshina, K. (2017). What is a Good Secure Messaging Tool? The EFF Secure Messaging Scorecard and the Shaping of Digital (Usable) Security. *Westminster Papers in Communication and Culture*, *12*(3), 51–71. doi:10.16997/wpcc.265

Nielsen, J. (2012). Usability 101: Introduction to usability. Retrieved November 3, 2019, from https://www.nngroup.com/articles/usability-101-introduction-to-usability/

Niinimäki, T. & Lassenius, C. (2008). Experiences of instant messaging in global software development projects: A multiple case study. In *2008 IEEE international conference on global software engineering* (pp. 55–64). doi:10.1109/ICGSE.2008.27

Ou, C. X. J., Davison, R. N., Liang, Y., & Zhong, X. (2010). The Significance of Instant Messaging at Work. In *Proc. Fifth Int. Conf. Internet and Web Applications and Services* (pp. 102–109). doi:10.1109/ICIW.2010.23

Ou, C. X. J. & Davison, R. M. (2010). The impact of instant messaging in the workplace. In *AMCIS 2010 Proceedings* (p. 136). Retrieved November 30, 2019, from http://aisel.aisnet.org/amcis2010/136

Ou, C. X. J., Davison, R. M., & Leung, D. (2014). Instant messenger-facilitated knowledge sharing and team performance. *International Journal of Knowledge Content Development & Technology*, *4*(2), 5–23. doi:10.5865/IJKCT.2014.4.2.005

Pahl, C. (2015). Containerisation and the PaaS cloud. *IEEE Cloud Computing*, *2*, 24–31. doi:10.1109/MCC.2015.51

Papazoglou, M. P. & Dubray, J. J. (2004). *A survey of web service technologies: Technical report dit-04-058*. University of Trento. Trento (Italy), Via Sommarive 14.

Patil, S. & Kobsa, A. (2004). Instant messaging and privacy. In *Proceedings of HCI* (Vol. 4, pp. 85–88).

Pearce, M., Zeadally, S., & Hunt, R. (2013). Virtualization: Issues, security threats, and solutions. *ACM Computing Surveys (CSUR)*, *45*(2), 17. doi:10.1145/2431211.2431216

Pi, S.-M., Liu, Y.-C., Chen, T.-Y., & Li, S.-H. (2008). The influence of instant messaging usage behavior on organizational communication satisfaction. In *Proceedings of the 41st annual hawaii international conference on system sciences (HICSS 2008)*. IEEE. doi:10.1109/hicss.2008.445

Porter, T. & Gough, M. (2007). Architectures. In T. Porter & M. Gough (Eds.), *How to cheat at VoIP security* (pp. 45–110). How to Cheat. Burlington: Syngress. doi:10.1016/B978-159749169-3/50004-9

Priologic Software Inc. (2019). EasyRTC. Retrieved October 10, 2019, from https://easyrtc.com/

Quan-Haase, A. (2010). Self-regulation in instant messaging (IM): Failures, strategies, and negative consequences. *International Journal of e-Collaboration (IJeC)*, *6*(3), 22–42. doi:10.4018/978-1-61350-459-8.ch009

Ramadan, H. H. & Kashyap, D. (2010). Quality of service (QoS) in cloud computing. *International Journal of Computer Science and Information Technologies (IJCSIT).*

Ramsdell, B. & Turner, S. (2019). RFC 8551: Secure/multipurpose internet mail extensions (S/MIME) version 4.0 message specification. *Internet Engineering Task Force.* Retrieved from https://tools.ietf.org/html/rfc8551

Reed, A. H. & Knight, L. V. (2010). Effect of a Virtual Project Team Environment on Communication-Related Project Risk. *International Journal of Project Management*, *28*(5), 422–427. doi:10.1016/j.ijproman.2009.08.002

Rennecker, J., Dennis, A. R., & Hansen, S. (2006). Reconstructing the stage: The use of instant messaging to restructure meeting boundaries. In *Proceedings* *of the 39th annual hawaii international conference on system sciences (HICSS'06)* (Vol. 1, 27a–27a). doi:10.1109/HICSS.2006.411

Rennecker, J. & Godwin, L. (2003). Theorizing the unintended consequences of instant messaging for worker productivity. *Sprouts: Working Papers on Information Environments, Systems and Organizations*, *3*(3), 137–168. doi:10.1.1.304.9410

Resig, J. & Teredesai, A. (2004). A framework for mining instant messaging services. In *Proceedings of the 2004 Siam DM conference.*

Rittinghouse, J. W. & Ransome, J. F. (2005). *IM Instant Messaging Security.* Elsevier Science. doi:10.1016/B978-1-55558-338-5.X5000-0

Rosenberg, J. (2013). RFC 6914: SIMPLE made simple: An overview of the IETF specifications for instant messaging and presence using the session initiation protocol (SIP). *Internet Engineering Task Force.* Retrieved from https://tools.ietf.org/html/rfc6914

Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., . . . Schooler, E. (2002). RFC 3261: SIP: Session initiation protocol. *Internet Engineering Task Force.* Retrieved from https://tools.ietf.org/html/rfc3261

Saint-Andre, P. (2004). RFC 3920: Extensible messaging and presence protocol (XMPP): Core. *Internet Engineering Task Force.* Retrieved from https://tools.ietf.org/html/rfc3920

Saint-Andre, P. (2005). Streaming XML with Jabber/XMPP. *IEEE internet computing*, *9*(5), 82–89. doi:10.1109/MIC.2005.110

Saint-Andre, P. (2009). RFC 3921: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. *Internet Engineering Task Force*, *23*. Retrieved from https://tools.ietf.org/html/rfc3921

Saint-Andre, P., Smith, K., & TronCon, R. (2009). *XMPP: The definitive guide: Building real-time applications with jabber technologies* (1st ed.) (O. Media, Ed.). O'Reilly Media. ISBN: 059652126X.

Salovaara, A. & Tuunainen, V. K. (2013). Software developers' online chat as an intra-firm mechanism for sharing ephemeral knowledge. In *34th international conference on information systems (ICIS 2013).*

Schollmeier, R. (2001). A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Proceedings first international conference on peer-to-peer computing* (pp. 101–102). IEEE. ISBN: 0-7695-1503-7.

Scholten, B. (2007). Integrating ISA-88 and ISA-95. In *ISA Expo*. Retrieved November 30, 2019, from https://www.isa.org/pdfs/integrating-isa-88-and-isa-95/

Sergiienko, A. (2014). *WebRTC Blueprints*. Packt Publishing Ltd. ISBN: 978-1783983100.

Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, *3*(5), 637–646. doi:10.1109/JIOT.2016.2579198

Sit, E. & Morris, R. (2002). Security considerations for peer-to-peer distributed hash tables. In *International workshop on peer-to-peer systems* (pp. 261–269). Springer. doi:10.1007/3-540-45748-8_25

Steinmetz, R. & Wehrle, K. (2005). *Peer-to-peer systems and applications*. Lecture Notes in Computer Science. Springer Berlin Heidelberg. ISBN: 9783540320470. Retrieved from https://books.google.de/books?id=nqEMBwAAQBAJ

Sugano, H., Day, M., & Rosenberg, J. (2000). RFC 2778: A model for presence and instant messaging. *Network Working Group*, (2778). doi:10.17487/RFC2778

Sugerman, J., Venkitachalam, G., & Lim, B.-H. (2001). Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. In *Usenix annual technical conference, general track* (pp. 1–14). ISBN: 1-880446-09-X.

Tidwell, D. (2000). Web services-the web's next revolution. *IBM developerWorks*. Retrieved December 1, 2019, from https://docplayer.net/9431715-Web-services-the-web-s-next-revolution.html

Unger, N., Dechand, S., Bonneau, J., Fahl, S., Perl, H., Goldberg, I., & Smith, M. (2015). SoK: Secure Messaging. In *2015 IEEE Symposium on Security and Privacy* (pp. 232–249). IEEE. doi:10.1109/SP.2015.22

van Eyk, E., Iosup, A., Seif, S., & Thömmes, M. (2017). The SPEC cloud group's research vision on FaaS and serverless architectures. In *Proceedings of the 2nd international workshop on serverless computing* (pp. 1–4). ACM. doi:10.1145/3154847.3154848

van Eyk, E., Toader, L., Talluri, S., Versluis, L., Uta, A., & Iosup, A. (2018). Serverless is more: From PaaS to present cloud computing. *IEEE Internet Computing*, *22*(5), 8–17. doi:10.1109/mic.2018.053681358

Wang, L., Tao, J., Kunze, M., Castellanos, A. C., Kramer, D., & Karl, W. (2008). Scientific cloud computing: Early definition and experience. In *2008 10th IEEE international conference on high performance computing and communications* (pp. 825–830). Ieee. doi:10.1109/HPCC.2008.38

Wang, L., Von Laszewski, G., Younge, A., He, X., Kunze, M., Tao, J., & Fu, C. (2010). Cloud computing: A perspective study. *New Generation Computing*, *28*(2), 137–146. doi:10.1007/s00354-008-0081-5

Williams, N. & Ly, J. (2004). *Securing public instant messaging (IM) at work* (tech. rep. No. 040726A). Swinburne University of Technology. Australia.

Wu, C., Liang, H., Chiu, S. M., & Yuan, C. (2017). A study of impact of instant messaging on job performance through employee empowerment. In *2017 portland international conference on management of engineering and technology (PICMET)* (pp. 1–10). doi:10.23919/PICMET.2017.8125343

XMPP Standards Foundation. (2018). XEP-0373: OpenPGP for XMPP. Retrieved July 17, 2019, from https://xmpp.org/extensions/xep-0373.html

Yadav, T. & Rao, A. M. (2015). Technical aspects of cyber kill chain. In *International symposium on security in computing and communication* (pp. 438–452). Springer. doi:10.1007/978-3-319-22915-7_40

Yang, B. B. & Garcia-Molina, H. (2004). Designing a super-peer network. In *Proceedings 19th international conference on data engineering (cat. no.03ch37405)*. IEEE. doi:10.1109/icde.2003.1260781

Yi, S., Li, C., & Li, Q. (2015). A survey of fog computing: Concepts, applications and issues. In *Proceedings of the 2015 workshop on mobile big data* (pp. 37–42). ACM. doi:10.1145/2757384.2757397