MASTERARBEIT

Titel der Masterarbeit
“Interconnected Virtual Organizations based on a Novel Data-Oriented Architecture”

verfasst von
Christian Vorhemus BSc

angestrebter akademischer Grad
Diplom-Ingenieur (Dipl.-Ing.)

Wien, 2015
Studienkennzahl lt. Studienblatt: A 066 926
Studienrichtung lt. Studienblatt: Masterstudium Wirtschaftsinformatik
Betreut von: Univ.-Prof. Dipl.-Ing. Dr. Erich Schikuta
First of all, I want to thank my adviser Dr. Erich Schikuta for his support and guidance. The process of arranging loose ideas to a coherent topic was challenging, especially because the field of distributed systems is wide and extensive. When the first concrete concepts were born, he helped me to do not lose focus on the purpose and the use cases of the main paradigms presented in this thesis. Furthermore, I had free reign to realize my ideas and room to make mistakes which was an important experience and lesson for me.

Second, I want to thank my parents Roland and Elfriede Vorhemus for their support during my entire life. A lot of things I achieved would not have been possible without their help. I appreciate that I had a large area of freedom in the decisions I made, starting with the choice of study, the decision to study abroad or finishing this thesis in another country. Even in my spare time they were there for me when I needed a safe ride home even late in the evening. This is not something to be taken for granted, it deserves appreciation and recognition.

Third, I owe my special gratitude to the Hilti Corporation and in this context especially to Dr. Martin Nemetz for his offer to take part in the IT Garage program and the possibility to realize the theoretical concept of Data Oriented Architectures for Hilti tools. Moreover my personal mentor Kristjan Gomboc was a great help to get devices for experiments, interviews with people in the R&D department and more concrete examples how Data Oriented Architectures could be used in a Hilti-related context.

Last but not least I want to express my thanks and recognition to my grandfathers Franz Vorhemus and Franz Langsam which have been an inspiration for me in many fields. My interests for technical and creative design originate from their passion for craft work. Being able to create something with own hands is a fantastic feeling, this is also the case for programming applications and algorithms that make our daily life easier. These interests are certainly a main reason why I have chosen Computer Science as my field of study, it is great to be part of a science that has already deeply influenced the world and has still enough potential to revolutionize the world again.
Abstract

The web has added a new dimension to the gap between the first world and the developing world. We have to start talking about a human right to connect.

Tim Berners-Lee

In the last 20 years, the Internet has started a triumphant march that has reached all corners of our world. The possibility of sharing information with everyone in the world in milliseconds was the basis for a whole new ecosystem of applications, distributed data and global collaboration platforms. Connecting people is still one of the main goals of the Internet, however, a rising interest exists to expand the technologies tailored for H2H (Human-to-Human) communication and enable more autonomous and smart M2M (Machine-to-Machine) communication to support our daily life in the background. Internet-of-Things (IoT) is a promising concept to make this possible but there are a lot of competitive technologies in the field that are not always compatible with each other. Furthermore, most of the connected systems in the Internet still follow the classic client-server-oriented architectural style while the environmental conditions have changed tremendously in the last years. More and more people use mobile devices that have more computing power and storage than advanced high performance computers 30 years ago. People nowadays are not passive consumers but actively produce content and share it with their loved ones. There is a (not fully developed but still growing) awareness for privacy, net neutrality and the so called "democratization of the web". The classic approach of degrading users to passive consumers while every active task is done on (web)servers does not longer reflect the true situation - sharing data and using resources around us should not only be achieved by using the infrastructure of large server providers but can also happen among users directly.

In this thesis, a new architectural style is presented, that fits the need of a modern and contemporary architecture and gives the users the possibility of exchanging data with others without being depended of infrastructure offered by companies. The architectural style follows several main principles, for example, data is not identified by its location but by the application which created it. Furthermore data should be split in small pieces (called data chunks) that use a well-known format and can be used directly by any other user. Before the architectural style called DOA (Data-Oriented Architecture) is described more detailed, an overview about the challenges that arise when multiple connected computers talk to each other is given. Furthermore a comparison of DOA with existing approaches is given and a practical implementation of the architectural style is shown. In the end, DOA is evaluated by qualitative and quantitative criteria.
Das World Wide Web hat eine neue Dimension zwischen der Ersten Welt und der Dritten Welt geschaffen. Wir sollten über ein "Menschenrecht zur Vernetzung" nachdenken.

Tim Berners-Lee

## Contents

| List of Figures | xi |
| List of Tables | xii |

### 1 Introduction

1.1 Motivation ........................................ 1  
1.2 Definition of architectural styles .................. 6  
1.3 A service-oriented and a data-oriented view .......... 7  
1.4 Application scenarios for DOA ..................... 10  
1.5 Terminology ....................................... 13  
1.6 Purpose and boundaries of this thesis ............... 14

### 2 Challenges for Interconnected Entities

2.1 Interoperability and basic data exchange ............ 17  
2.2 Data Management ................................... 21  
2.3 Heterogeneity and Fairness .......................... 27  
2.4 Scalability ......................................... 30  
2.5 Security ............................................ 31

### 3 Data Oriented Architecture

3.1 Design goals ....................................... 35  
3.2 Addressees ......................................... 39  
3.3 Architectural style .................................. 39  
3.3.1 Classification .................................. 39  
3.3.2 System model .................................... 40  
3.3.3 Design Principles ............................... 41  
3.3.4 Vocabulary ...................................... 48  
3.3.5 Design rules ..................................... 51  
3.3.6 Analysis ......................................... 52  
3.4 Virtual Organizations ............................... 53  
3.5 DOA-SOA binding .................................... 56

### 4 Comparison with Existing Approaches

4.1 Historical development ............................... 57
List of Figures

1.1 Disk and memory size and price through time ............................................... 3
1.2 Classic SOA paradigm .................................................................................... 8
1.3 Data-Oriented Architecture paradigm ............................................................. 9
2.1 Example scenario for channel hopping communication initiation ...................... 21
2.2 Data access unification .................................................................................. 23
2.3 RDF semantic integration ............................................................................ 25
2.4 Security characteristics of P2P networks ....................................................... 33
3.1 DOA technology stack .................................................................................. 40
3.2 DOA from a business perspective .................................................................. 41
3.3 Data-oriented consumer application approach ................................................ 43
3.4 Data chunk sections ..................................................................................... 46
3.5 DOA Data Access layer ................................................................................ 47
3.6 Data sharing process in a DOA based environment .......................................... 55
3.7 DOA-SOA binding ........................................................................................ 56
5.1 DOA Class Diagram ..................................................................................... 69
5.2 DOA Garbage Collection Part 1 ..................................................................... 70
5.3 DOA Garbage Collection Part 2 ..................................................................... 71
5.4 DOA at the construction site ........................................................................ 74
5.5 Real-time Tool Monitor System based on DOA ............................................. 76
5.6 Deployment model ........................................................................................ 76
5.7 Activity diagram ........................................................................................... 77
6.1 Measurement setup ....................................................................................... 84
6.2 Message diffusing time for 2-5 nodes ............................................................. 85
6.3 Bluetooth/Wifi-direct battery usage ............................................................... 86
6.4 Messages passed in the network ................................................................... 87
7.1 Example of two entities and their bloom filter ............................................... 89
## List of Tables

1.1 Number of devices connected to the Internet ............................................. 2  
1.2 Overview over architectural styles ......................................................... 10  
2.1 Wireless technologies ............................................................................. 19  
2.2 Data models ......................................................................................... 23  
3.1 DOA Vocabulary ..................................................................................... 51  
4.1 Comparison of DOA with other approaches ............................................. 61
CHAPTER 1

Introduction

1.1 Motivation

Parts of this thesis were written on long train journeys. A train is a wonderful way of travelling, not only because it allows finishing some work but also because of the people in it. Even if they are strangers, all have a common goal: Reach a certain destination. A train is a temporal gathering of people from different background, with different professions and maybe also from different countries and it is not false to assume that some of them are experts in a special area. I once met a professional chess player on a train and had a discussion about the most aggressive opening strategies. If I had to write a paper about aggressive opening strategies in chess at that time, this man would have been an excellent primary source for me. Of course, one must be very lucky to meet a stranger on the train that is an expert in a field that is of high interest for oneself at the moment. But the question I asked myself is: Couldn’t we invent a system that helps people to find strangers in their local domain (the train) that have similar interests? What if an expert of architectural styles that could have given precious inputs for this thesis was sitting just a few rows in front of me and I had no clue? And if we go further, we can not only include people themselves but also resources they have. Maybe, someone in the train is trying to stream a video but Internet connection is not fast enough. What if someone else is willing to share some bandwidth of his phone temporarily and gets paid for it? Not only the train, the world is full of people with inquiries and other people that could fulfil this demand in the local environment, but the problem is, that they often do not find each other.

The starting point of our journey through the world of connectivity is the variety of devices. In today’s world, interconnection among more and more different types of entities is still proceeding. While in the early days of the Internet the classification of entities was two-tiered
<table>
<thead>
<tr>
<th>Category</th>
<th>2013</th>
<th>2014</th>
<th>2015</th>
<th>2020</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automotive</td>
<td>96.0</td>
<td>189.6</td>
<td>372.3</td>
<td>3,511.1</td>
</tr>
<tr>
<td>Consumer</td>
<td>1,842.1</td>
<td>2,244.5</td>
<td>2,874.9</td>
<td>13,172.5</td>
</tr>
<tr>
<td>Generic Business</td>
<td>395.2</td>
<td>479.4</td>
<td>623.9</td>
<td>5,158.6</td>
</tr>
<tr>
<td>Vertical Business</td>
<td>698.7</td>
<td>836.5</td>
<td>1,009.4</td>
<td>3,164.4</td>
</tr>
</tbody>
</table>

Table 1.1: Number of devices connected to the Internet

(server and client) and each tier had a clearly assigned purpose (servers were centralized points of accessible services and data while clients were access points for this services) today these hard boundaries are becoming ever less distinct. One of the first attempts to break this two-tiered architecture style were Peer-to-Peer (P2P) networks. Bar00 simply defines P2P as “the sharing of computer resources and services by direct exchange”. In a P2P network, each participant is treated equally and is both, requester and supplier of services or data at the same time. P2P networks are usually more flexible, scalable and failure-resistant than client-server oriented ones since the server as a single point of failure and a bottleneck is spared out.

The reasons why client-server architectures were developed and used before P2P architectures are manifold. Bar00 claims that client-server architectures were simpler to use. Furthermore, for an efficient use of a P2P network, a critical mass of computer users had to be exceeded. This is especially true in case of file sharing networks such as Napster or Gnutella TH06 which were one of the first usage scenarios of P2P networks. It’s a trivial requirement that data must be present on peers before it can be found by other peers. Another requirement is that communication bandwidth between all peers is large enough and all entities are stable and have enough storage and memory capacity to process the amount of data before content can be moved from one peer to another. It can be observed that three essential factors related to the statements just given have changed tremendously in the last years. Those are:

1. The number of devices connected to the Internet is increasing.
2. The quality of computer equipment (size of storage, size of memory, etc.) is increasing while the price for this components is decreasing.
3. The amount of data which is produced daily is increasing.

Table 1.1 shows the number of devices connected to the Internet (in millions) sorted by area of application according to Gartner Gar. Especially in the consumer sector the number of devices connected to the Internet increased by about 56%. Furthermore, Gartner expect a growth to 13 billion in 2020.
The conclusion that can be drawn from this progress is that the number of devices that have the ability to communicate with each other is increasing. This implies that the underlying technology and the components used in computers and mobile peers is getting more advanced. Furthermore, the price for these components is decreasing which boosts the distribution of these devices. Figure 1.1 gives a justification for this assumption. As it can be seen in the right part, the average memory size increased from about 65 Megabyte (2000) to 8 Gigabyte (2014) [MDP]. The costs of memory bars had a peak in the years 2005 and 2006 but they are now lower than in the 2000’s. In the left part, the average storage size for a hard disk drive was constantly increasing from 40 Gigabyte (in 2000) to 3 Terabyte (in 2014) [DDP]. In the same time, the price was falling from around 300 USD in the early 2000’s to around 100 USD in 2015.

Since storage has become cheaper and hard disks got a bigger storage capacity, it’s not a surprise that the amount of data stored worldwide has also increased. This is also driven by consumers since they heavily use data-intensive applications such as video platforms, social networks or image sharing sites. According to [CSC], the amount of data produced today is about 7.9 Zettabyte \((7.9 \times 10^{21} \text{ Bytes})\) and will increase to 35 Zettabyte in 2020. More than 1/3 of the data produced will be stored in "the cloud".

The "smartification" of objects in our everyday life has reached phones, watches, cars and will reach home appliance and the organism of humans or animals as well. A Dutch company called "Sparked" has invented an implant for cows which measures vital signs such as body temperature or heart rate [GUA]. The biological information of each cow is than transmitted wirelessly to a central server where the data is evaluated. The founders expect to gain information about the herd which could be useful for the farmer such as: Is a cow pregnant? Is a cow ill and could it infect other cows? It can also be measured how much milk a cow is giving and an additional GPS sensor monitors the movement of the animal. It may seems that this project
is only interesting for the owner of the cows, however, there is also room for environmental research. Especially the combination of environmental data (e.g. air pollution) and the impact for organisms is an ambitious field of study. In "Nature Education Knowledge", the process of measuring environmental impacts by bioindicators was described [HMT11]. Bioindicators are organisms and their measurable attributes such as biochemical processes or body temperature. For example, it is known that some trouts have a thermal tolerance of 20°–25°C which means the presence or absence of these trouts can be used as a bioindicator for water temperature. In a study by Hasselbach et al. (2005), the impact of heavy metals in the ecosystem caused by a zinc mine was analysed [HVHF+05]. They used a moss called "Hylocomium splendens" as an indicator for the metal pollution since mosses act very sensitive to this form of contamination.

The prerequisite for this research is that data about organisms and the environment is available for a certain period of time. In wild nature, it may be likely that a constant Internet supply is not given. This raises the question how data is collected and how data is stored. In this case, "sensor networks" are used often. Sensor networks consist of small and wirelessly connected nodes which collect and share data. In some cases, some nodes may have additional functions such as transmitting data to a base station but typically all nodes are treated equally. In fact, sensor networks are mostly peer-to-peer networks and therefore enjoy the advantages of robustness and better scalability. In a sensor network, nodes are often distributed over a large field and therefore, only a few nodes have access to a base station which may collect data or send it to a central server. This means, nodes may have to pass data to other nodes which are closer to the base station since they are not in transmission range to the base station. In case of a change of environmental conditions, a subset of nodes may be cropped from the rest of the network. Then, the question arises if these separated nodes can still continue their work and automatically restore a stable network when the connection to the other nodes is recovered again. If we look at the world today and all connected devices, we may come to the conclusion that we move towards a global sensor network consisting of multiple different entities. Smartphones and other smart devices are constantly collecting data through various sensor such as GPS, gyroscope or compass. This data chunks are further processed by applications or web services such as traffic information services. These services collect GPS data from multiple users, aggregate them and use it to give a real time picture of the current traffic condition on the streets worldwide. If velocity of devices is known, sophisticated algorithm are also able to predict traffic flows to a certain degree.

Today, most of these services are placed on centralized web servers and clients access them via APIs. So we still face a master-slave environment like in the early days of connected computers. A reason to push the development of distributed systems away from client-server architectures towards decentralized networks should limit the power of big service providers. Originally,
the concept of "cloud computing" meant that resources are anywhere and can be accessed within a network \cite{KTMF09}. Cloud computing today does not completely meet this description since consumers contact specific cloud computing providers and resources are consumed only from this provider. Even if data centers of these providers are typically failure resistant and stable since multiple distributed computers are used, consumers make themselves dependent on single companies with several disadvantages (e.g. vendor lock-in). The question arises, if customers should put their data exclusively in the hand of big service providers and relinquish all rights on their personal data. This is not only an issue of privacy but also an economic one, since some contracts between consumers and service providers entitle the providers to make money out of the content a consumer has produced \cite{MAS}. Evgeny Morozov has suggested that data produced by consumers should not be a commodity that is hold only by service providers rather than a "public good" to which everyone has access to \cite{MOR}. Critics replied that the publication of this data chunks would be an even deeper attack on privacy. However, this is only the case when data chunks can be clearly assigned to the user who produced it. The value of data emerges by the linking of multiple chunks of data, there is hardly any economic use for single chunks of data (e.g. timestamps without further reference, GPS tags, etc.) which cannot be assigned to other chunks.

Another reason why the exclusive use of client-server architectures is not far-sighted relies on the characteristics of computers in general which have changed tremendously. Today, clients like smartphones get more and more computing power, storage size is increasing and the total number of smartphones and other connected devices used worldwide is getting higher as well. In \cite{FMK10}, a group of 222 smartphone users showed a minimum daily usage time of 4 and a maximum of 200 minutes while the phone was switched on the entire time. This means, most of the time, the computing power of the phone lies idle. There are already attempts to use these fallow capacities for scientific projects. One of them was "Power Sleep", an application for smartphones which can be used for deciphering protein sequences \cite{GIZ}. Small data chunks are sent to smartphones of users which have installed this app, the phones make the calculations and send the results back to the servers of the University of Vienna. Taking one step further, in the future, devices of users may collaborate automatically and share unused resources. By establishing a payment system, users may also sell resources they don’t need. In all these cases, it is more efficient when devices can talk directly to each other without the need of additional servers. When devices communicate directly with each other over channels such as Bluetooth or Wi-fi direct, implicitly additional information about them can be derived, namely that they are in transmission range to each other. They have in common that both share the same location. If one of the devices knows in which environment it currently is, it may can conclude that the user of other devices near them may have similar interests. For example, if researchers
join a conference about wireless communication, it is likely that all of them have at least one similarity, namely their interest in wireless communication. Smartphones of these researchers could exchange information about them automatically (e.g. contact information or previous publications). If users are in a shopping center, devices may automatically exchange information about discounts and advertisement boards show customers specific advertisement when someone passes by. Users on the street may share traffic information, information about free parking spots or warns of traffic jams or accidents. There are a lot of use cases for this direct form of communication whereby in some cases it makes perfect sense to use client-server architecture as well. At this point it is noteworthy to say that the usage of a client-server architecture and a P2P architecture is not mutually exclusive. Both architecture styles can be used concurrently.

Since the understanding and classification of the term "architecture style" is critical for the assignment of the data-oriented architecture presented in this thesis, the next sub-chapter tries to give a distinct definition of what architectural styles are.

1.2 Definition of architectural styles

In literature, the terms "architectural style" and "architectural pattern" are not always clearly separated and used as synonyms in some publications. In general, the terms are always related to the term "software architecture". [Fie00] defines a software architecture as an "abstraction of the run-time elements of a software system during some phase of its operation. A system may be composed of many levels of abstraction and many phases of operation, each with its own software architecture". The principle of abstraction is a commonly used concept whenever complex systems should be described. [Kra07] asked if abstraction is the key to computing and answered the question with yes. He claims that abstraction is necessary to master complexity, he also says that "Abstraction skills are essential in the construction of appropriate models, designs, and implementations that are fit for the particular purpose at hand. Abstract thinking is essential for manipulating and reasoning about abstractions [...]". Abstraction is also an essential part when defining the term "architectural style". For [MKMG96] an architectural style is a family of systems that share the same underlying computational model. On the contrary, according to [PKG+00] architectural patterns "describe reusable and extensible technical solutions to common design problem". So the main difference between those two concepts is that architectural styles are more abstract while architectural patterns are more specific and designed for a smaller use case. Following this definition, a client-server architecture or a P2P architecture would be classified as an architectural style since these styles are just a basic underlying concept to structure and organize the interaction between components while for example the Model-View-Controller (MVC) concept would be treated as an architectural pattern since it solves the
The problem of organizing software applications on a lower abstraction level.

The classification of SOA is harder since on the one hand it is a basic concept how software can be organized, on the other hand it solves the specific task of incompatible autonomous software by introducing a coherent interface. W3C defines SOA as "A set of components which can be invoked, and whose interface descriptions can be published and discovered" \[W3C\]. For OASIS, SOA is a "paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains" \[MLM*06\]. Both definitions see SOA as an abstract way to organize components, therefore they support the opinion that SOA should be classified as an architectural style.

The distinction between architectural styles and architectural patterns is necessary since both concepts have different fields of application. The more abstract the concept, the more manifold the use cases (the client-server architecture for example has a broader area of application than the MVC concept). Moreover, architectural styles typically provide the following things \[MKMG96\]:

1. A vocabulary. This is a list of components that are used in the architectural style.
2. Design rules. These rules define how components are related to each other.
3. Semantic interpretation. This defines the meaning behind the vocabulary and the design rules.
4. Analyses. This includes any methods to analyse the style, for example a deadlock detection system for client-server message passing.

In this thesis, a new architecture style will be defined. The four elements (vocabulary, design rules, semantic interpretation, analyses) will be used to structure the style.

1.3 A service-oriented and a data-oriented view

The shift from existing mutually not compatible software systems to a Service-Oriented landscape was a big trend in recent years. From a business perspective, SOA was intended to make IT systems more flexible since small autonomous software systems can be orchestrated to more complex ones. These flexible business processes should lead to a shorter "Time to Market", furthermore SOA should help to save costs in the medium and long term. From an IT architecture point of view, SOA should increase the transparency of the IT system due to the "divide and conquer" technique. Standardized APIs should increase the reusability of software parts and lead to a better maintainability.
It has become evident that these promises have only been fulfilled partially. Some studies show a fail rate of about 50% when SOA was implemented in businesses [TEC]. "The service is only built for one application and it’s never going to be used again" claimed Anne Thomas Manes, vice president of Burton Group. The question if SOA is dead was raised multiple times in the IT community. Jason Bloomberg, author and journalist for Forbes magazine said that "SOA as a technology has already died due to Cloud Computing and due to the intrinsic complexity of Web services" [INF]. One reason for this pessimistic attitude towards SOA nowadays was the early hype for this architectural style. Companies primarily saw SOA as a way to save costs without great efforts. Furthermore, introducing a service layer on autonomous entities to get a standardized interface was more complex than expected. The question that arises is, if the focus on services is useful for every use case. When autonomous entities offer services for others, they hide complexity since only data that is provided by the service can be explored. This means the service provider and the service requester must agree on what data is needed. When service requester and service provider are from different domains or companies, usually the service requester has to follow the structure and the service provider specifies. In most cases, the service requester will have to convert or rework the response of the service provider since the provided service will not always deliver exactly the output the service requester needs. It is possible to bypass this problem by lowering the abstraction level and go to data level. This increases transparency since single data chunks are considered, but it completely outsources the process of handle and manipulate data to the consumer. Similarly to SOA, this new approach can be used to structure software systems but focuses more on data and therefore it should be named "Data Oriented Architecture" (DOA).

In contrast to SOA, DOA has a broader area of application. While SOA is used to increase interoperability between autonomous software parts and primarily focus on IT organization, DOA has multiple goals:
• Reduce redundancy. Since data chunks are shared among applications and users, they only have to be stored once and can be used by multiple applications or users.

• Avoid vendor lock-in. Data chunks can be shared directly among users without centralized servers. This decreases dependency from single service providers.

• Data can be treated as a public good. Users can decide which data should be kept private and which should be made public. Public data is available for everyone and can be used for any purposes.

• Provide a framework for modern architectures. As stated above, trends like increasing number of Internet users, more powerful hardware and an increasing number of data can be observed, however, most of today’s underlying system architectures are designed for the "old" Internet. For example, when the number of data sources on the Web is small, it is easy to identify which sources are trustworthy and which are not. When the number of Internet users is increasing (especially if the number of private providers of data is increasing), it is harder to identify trustworthy sources. This becomes even more evident, when data is not provided on servers but shared among users. In this case, traditional reputation concepts (e.g. Google’s PageRank Algorithm) are not working, however, some concepts may be borrowed from P2P systems.

Figure 1.2 shows an abstract visualization of a SOA system. Autonomous entities offer services that can be used by consumers while the logic of the entities is hidden. Consumers can only use those methods, that are offered by the service which means they are bound to the output of the service provider. In contrast, figure 1.3 shows a visualization of a data-oriented system. Entities can store chunks of data in a pool which is shared among the users. This pool is actually just an abstract depiction since data chunks remain on the devices of the users but are public accessible and developers can use these data chunks to build up applications based on them.
<table>
<thead>
<tr>
<th>Category</th>
<th>Architecture styles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication</td>
<td>Service-Oriented Architecture, Message Bus</td>
</tr>
<tr>
<td>Deployment</td>
<td>Client-Server, Peer-to-Peer, N-Tier</td>
</tr>
<tr>
<td>Domain</td>
<td>Domain-Driven Design</td>
</tr>
<tr>
<td>Structure</td>
<td>Blackboard system, Data-Oriented Architecture</td>
</tr>
</tbody>
</table>

**Table 1.2: Overview over architectural styles**

Table 1.2 shows a list of commonly used architecture styles ordered by their main area of application \[\text{MST}\]. While SOA mostly concerns communication and is easy to classify in this scheme, DOA is a structural style. Similar to a blackboard-based system where a set of "experts" share a so called "blackboard" and modify data on this pool of knowledge, DOA structures systems by sharing data. DOA is especially useful when a lot of data is produced by independent sources (e.g. smartphones) and when these sources are powerful enough (e.g. data storage, memory size, ...) to handle data chunks without the need of centralized sources.

### 1.4 Application scenarios for DOA

In the following, a non-exhaustive list of possible use case scenarios for DOA is given.

- **Automatically build up virtual organizations to perform customer-centric tasks**
  
  Language is one of the most powerful human tools. It enables us to express opinions and talk about abstract ideas, it can be used to warn others and it can be used to share experience and knowledge with others. Even when humans do not communicate actively, they have the ability to passively listen to others. Two fundamental things are necessary to make this concept work: The persons must be close to each other in the sense of geographical distance and they must speak the same language. Since all our senses are constantly collecting "data" from our environment, our brain has to filter all the things that do not seem to be relevant for us. This outstanding ability enables us to remember important things for a long time but forget unimportant things almost immediately. It also helps us to set priorities and collect a coherent set of knowledge and things we care about. The research area that covers these topics is called Neuroscience, it is a mixture of different other sciences such as biology, psychology, linguistics, chemistry and also informatics. While this concept works quite well among humans, computers do not work that way. They do not search for other devices that may be interesting enough to share data with them. Computers have to be manually paired with others to communicate, they are not intended to communicate on their own. The idea of DOA is making this concept possible
through a uniform platform where devices can directly talk to each other without the need of further human interaction. When multiple people and their mobile devices are close to each other, the devices can build up a wireless ad hoc network (WANET) and form a Virtual Organization (VO). According to [KST10], a VO is "a nonphysical, communication model whose purpose is to achieve a common goal". This VO is only temporal and virtual which means, after the goal is reached, the organization disappears. An real-case example is sharing computational resources to others. For example, when a mobile device temporarily needs additional computational power to solve a specific task, it may ask other devices around them if they could share their (unused) resources. Devices that are willing to do so automatically build up a connection, the initiator device sends out jobs that have do be solved to other devices, they compute the solution and send the result back to the initiator. They may also decide to get a reward for this effort, additional payment or negotiation models can be implemented as well [MPS14]. This concept is much closer to "real" cloud-computing since anyone can be a contributor of resources and it is not clear from the outset, on which device the computational operations will be performed.

However, this concept raises several ethical, technical and legal questions. For example, how far should a person relinquish control to computer? What wireless technologies can be used to communicate and how can devices make sure that they speak the same "language"? How can security be ensured in such an environment? These and other questions are formulated more detailed in chapter 2.

- **Search for data chunks in the local neighborhood**

  Very often, people are searching for things in their surrounding. At a scientific conference, people may want to find discussion partners working in the same field of research. At the street, people may want to know if congestion is on their way home. In every day life people are looking for commodities others are offering. All of these use cases can be achieved by using a platform where people can share their profile, their knowledge about congestion at a specific street or offer commodities at eShops. However, this requires that people put time and effort to create such profiles. It is a core goal of DOA to automate this procedure. If people install social media applications upon a data oriented architecture, constantly information such as interests of a person are broadcasted. Which information gets public can be decided by the application, so it’s up to the user to install only trustworthy applications. If people are looking for a person with special skills they need right now (e.g. a craftsman), they can start a search request which will be broadcasted to other people in the surrounding. If the request gets to the device of a craftsman, a search response (e.g. the contact information) of the craftsman is sent back to the initial sender.
• Establish a PAN/LAN or mesh network in areas with no Internet connection

Webservices help us in different use cases. We use online storage, online messenger services and online collaboration tools. In situations where online access is not possible or very slow, DOA could help to automatically establish a network with peers in transmission range to sustain connection between them. Consider a situation where scientists start a expedition to a desert. At the base camp, Internet connection may be available, but when they leave the camp, they may also lose Internet connection. However when they want to analyse data of multiple sensors in real time, they would need to set up an ad hoc network and pair their devices to exchange data. DOA would simplify this procedure since chunks of data are automatically exchanged among peers within transmission range. The procedure of establishing data connection would work without the need of further human interaction.

• Trying to ensure eventual consistency in case of netsplit

The CAP theorem is one of the fundamental statements in relation to distributed computing system. It was formulated by Eric A. Brewer and it states that a distributed system can never achieve all of the three properties: Consistency, availability and tolerance to network partitions at the same time [BRE]. However it is possible that at least two of these three properties can be ensured simultaneously. A classic relational database for example ensures consistency and availability but it doesn’t ensure partition tolerance. Hence, if partitions or nodes in the network fail, the distributed system will not work correctly. Another example is Domain Name System (DNS) which ensures availability and partition tolerance but not consistency. This means, DNS servers will be available all the time and even if some of these servers become inoperative, the system will still work. However, this security comes at the expense of a lack of consistency, it may take days until changes of DNS entries have spread through the whole network. In this context, the term "eventual consistency" is relevant as well. It means that in a distributed system where eventual consistency is given, the most recent change of a data item will be returned to the requester eventually (even if not all nodes are in the same states yet) [Vog09]. This requires a conflict resolution strategy to clarify which node actually has the most recent value and which nodes hold outdated values. This is especially relevant in case of a netsplit, when parts of networks are suddenly cropped from the rest - in this case DOA could help to ensure eventual consistency. For example, when people want to withdraw money at cash dispensers, a connection is established to central servers to check the current account balance. If the network is disrupted, an error message is shown and no further interaction with the machine is possible. Developers could create an application based on the DOA paradigm that
could at least guide people to the nearest available cash dispenser that hasn’t run out of money yet. If a dispenser is empty again an error message will be shown and the machine is again not usable for the customer. With the DOA-based application, the cash dispenser could tell the current user if enough money is available and store this information on his mobile device. As the user goes through the city and passes other cash dispensers or users with the same application installed, the information which machines are still available and have enough money is spread through the network. This can help users to decide where the next workable cash dispenser is, even when some of them are disrupted from the network. This can be seen as a "support network" for the cash dispenser network.

1.5 Terminology

Below, a list of terms and their definition used in this thesis is shown. Some terms may not be defined clearly in literature, in this case, it was attempted to give a coherent and uniform definition.

- **Data Oriented Architecture (DOA)**: Data Oriented Architecture is a software architectural style with the purpose of simplifying the development of decentralized applications based on the principle that data should be broke into multiple small chunks of data and shared with the local environment automatically without the help of users.

- **Entity/Node/Peer/Device**: Basically all these terms refer to the same concept, they are just used for different contexts. An "entity" is an abstract description for the more concrete terms "node", "peer" and "device". An entity is an object that can stand for itself and plays a leading role within a certain context. If an entity is part of a mesh-network, it is called a "node". If an entity is part of a P2P network, it is called a "peer". If focus is set on the hardware characteristics of an entity, it is called a "device".

- **Resource**: In context of DOA, resources are goods that have value for others. A resource can either be physical (e.g. computer hardware), a service (e.g. offering network bandwidth) or immaterial (e.g. knowledge, data).

- **Virtual Organization (VO)**: There is no coherent definition of VOs in literature and it highly depends on the context how VOs are understood. For [KSI10], a Virtual Organization is "a nonphysical, communication model whose purpose is to achieve a common goal. It consists typically of a heterogeneous collection of people and organizations with respect to geographical limits and nature". While this definition is more abstract and focus on communication, for [FKT01] VOs are driven by common sharing rules since sharing is
“highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a virtual organization”. For DOA, both definitions are applicable but incomplete. On the one hand, DOA is more than a collection of sharing rules since it also provides mechanisms for data authentication. On the other hand, DOA is not an abstract communication model rather than a concrete architectural concept to implement applications that make use of decentralized data. Therefore, a more comprehensive definition should be given: A Virtual Organization is a temporal cooperation of at least two entities to achieve a common goal whereby the cooperation is driven by exchanging resources, data and knowledge.

- **Data Chunks**: A data chunk is an enclosed piece of information that can be interpreted by third parties and is independent from other chunks but can be chained with other data chunks to a data group. In context of DOA, data chunks are always key-value-pairs with header information.

### 1.6 Purpose and boundaries of this thesis

The scope of this thesis is limited to a selection of essential questions and a theoretic concept how DOA could answer these questions as well as a core implementation of DOA. In chapter 2, a survey about topics that are related to DOA is given, especially the Internet of Things, integration of data, linked data, authentication and security in P2P networks. Since the topic is extensive, only an overview about common problems and possible solutions should be given instead of a detailed elaboration of a specific question. This thesis should also rise awareness for the challenges in distributed system and may also be inspiration for other papers in this field.

It will further be discussed what issues need to be resolved before a data-oriented architecture as presented in this thesis can be built and questions are formulated that will be solved in chapter 3. A theoretical description of DOA as well as a sample practical implementation is given as well. From a practical point of view, distributed systems are usually harder to implement and debug than systems where components are only stored locally. The reasons are that developers have to deal with additional problems (e.g. network latency, temporal elusiveness of services or components, asynchronous event handling, etc.). It’s getting even harder when distributed systems are connected wirelessly since even more restrictions have to be taken into account (e.g. listening and sending simultaneously is usually harder or impossible on a wireless channel). For this reason, DOA offers the possibility to "outsource” the pain of wireless connections and exclude it from the application context. A DOA platform encapsulates the wireless connection
handler and offers interfaces for local applications to make use of data that was exchanged with other devices.

In the end, suggestions are made which technologies that already exist today can be used to make DOA work. It will also be discussed how existing technologies would have to be changed to make DOA more efficient. Performance measurements are limited to a small selection in this thesis.
Challenges for Interconnected Entities

When autonomous entities should communicate, a lot of questions have to be answered before. In this chapter, the most essential and basic questions are formulated that have to be resolved before the communication and data management process can take place. The questions concern 5 fields: Managing Interoperability, Data, Heterogeneity, Scalability and Security. In literature, these 5 topics are often named when it comes to the question which challenges in networking and IoT are the most relevant [BST11], [SNPR14]. In this chapter, technologies and solutions are presented that already exist but do not exactly fit for the requirements for collaboration between users as described in the previous chapter. For each topic, a research question is formulated. Data-oriented architecture should be able to cover these questions and provide possible solutions.

2.1 Interoperability and basic data exchange

A fundamental challenge for a cooperation of multiple entities is, that all of them have the ability to communicate with each other even when they have different backgrounds. Different backgrounds mean, that their communication process distinguishes from others which can rely on:

- Different communication channels: For example, some entities use wires for communication while others may want to communicate wirelessly.

- Different protocols: Even if entities share the same channel, it’s not clearly defined how a communication process starts or which messages should be exchanged.
Different policies: Even if all entities use the same protocol, some may be restricted to communicate only with specific other entities.

An essential question that arises here is: How do entities know on which channel someone will send messages? It would be easy to answer this, when entities could just make an agreement on which channel they will broadcast, but this would require that they are already able to communicate with each other. To solve this chicken-and-egg problem, typically standards are used. Standards are either provided by standardization institutes or are developed for specific business use cases by industrial research or within scientific communities. Standards usually also try to solve additional challenges that arise, for example the access of multiple entities to a shared infrastructure which requires multiplexing strategies. There are two types of channels that can be used when devices communicate with each other: Wired and wireless. Although wired communication is still feasible for many application scenarios, the increasing number of mobile devices has lead to a boost of further development of wireless technologies. The main advantage of wireless communication is - trivially - the lack of wires and therefore the reduction of costs and time to install and maintain wires. The second advantage is the simplicity for users. Multiple users can easily connect to wireless networks with a higher degree of mobility and less effort. According to [CIS] it is expected that the number of mobile connected devices will grow in the foreseeable future. Therefore and with consideration of the quoted advantages above, wireless communication tends to become the leading communication channel when we talk about the "Internet of Things".

While Ethernet has become the de facto standard for wired LAN, in wireless communication the spectrum of communication standards is still diverse. The reason for this is that the application domains for wireless networks are more multi-variant than for wired networks, the use of a technology depends on the transmission range, bandwidth, energy consumption, costs and mobility. Table 2.1 gives a comparison of frequently used wireless communication standards, their data rate, range and frequency [SSC07]. Which standard should be used depends on the application scenario. WiFi (IEEE 802.11) is a well developed and widespread standard which is supported by lots of devices. Its main advantage is the great community that actively enhance the standard (there is also a non-profit organisation called WiFi-alliance that certifies WiFi devices), multiple extensions for 802.11 have been developed such as 802.11n which supports MIMO (Multiple Input Multiple Output) [IEE]. A disadvantage of WiFi is its use of the 2.4 GHz band for which a licence is not necessary in most countries. This means that other devices may use the same frequency which can cause interference problem. Another more relevant disadvantage is energy consumption. As stated in [LSS07], the power consumption for Bluetooth and ZigBee is significantly lower than for UWB or WiFi. Bluetooth and ZigBee are intended for mobile
<table>
<thead>
<tr>
<th>Technology</th>
<th>WiFi 802.11n</th>
<th>ZigBee</th>
<th>WiMAX</th>
<th>Bluetooth</th>
<th>UWB</th>
<th>NFC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Wireless LAN, Internet</td>
<td>Sensor Networks</td>
<td>Metro Area Broadband Internet</td>
<td>Wireless Personal Area Network</td>
<td>Short-range indoor applications</td>
<td>Short distance device communication</td>
</tr>
<tr>
<td>Typical range</td>
<td>100m</td>
<td>70-100m</td>
<td>50km</td>
<td>1-100m</td>
<td>10-50m</td>
<td>0.2m</td>
</tr>
<tr>
<td>Data rate</td>
<td>108-660 Mbps</td>
<td>250 Kbps</td>
<td>75 Mbps</td>
<td>1-3 Mbps</td>
<td>400-1400 Mbps</td>
<td>500 Kbps</td>
</tr>
<tr>
<td>Frequency range</td>
<td>2.4 GHz</td>
<td>2.4 GHz</td>
<td>2.11 GHz</td>
<td>2.4 GHz</td>
<td>3-10 GHz</td>
<td>13.56 MHz</td>
</tr>
</tbody>
</table>

Table 2.1: Wireless technologies

devices and therefore designed to save energy.

When multiple standards exist, the question on which channel devices communicate is still not answered since they have to agree which standard should be used for a communication session before. In this case, human interaction can resolve this issue and pair devices manually, but this is not a satisfying solution when the communication process should be driven by automation. Three possible solutions are suggested:

1. Listening on multiple channels simultaneously for incoming messages.

2. Select one standard that must be used by all devices.

3. Detect and identify devices in the neighbourhood with other channels or sensor (e.g. cameras, electromagnetic sensors, ...).

In the first option, a device broadcasts inquiries on all possible channels. If other devices receive an inquiry, they respond according to the protocol specification. A simple visualization of this process is shown in figure 2.1. In this example, device A wants to scan for possible devices near them and broadcasts inquiries on all possible channels (the process is initiated on application level). If a Bluetooth module is available, the device sends inquiries on different frequencies to local other devices within transmission range. If concrete services are known that can be used, the Service Discovery Protocol (SDP) may be used. If a device receives this request, it sends a respond back to the initiator. If devices support multiple technologies, they may negotiate which channel should be used (this should be handled on application level as well).
Possible QoS parameters to decide which technology should be preferred can be energy consumption or transmission range. Negotiation mechanisms like [BVS13] that were constructed for service negotiation can be adopted for this use case.

However, this method has several disadvantages. First, it is time- and energy-consuming when all possible channels have to be monitored. Second, the broadcasting and listening process has to be repeated continuously to detect new devices that have come into transmission range while others may have left. For mobile devices, energy expenditure would be too high for practical usage, moreover a persistent scanning for devices of multiple devices close to each other may increase the probability of interference.

The second method is slightly simpler than the first one. It states that one standard should be selected that must be used by all entities that want to form a network. This obviates the need for channel selection and is therefore easier to implement. The obvious disadvantage of this method is a worse adaptability to device-centric use cases. For example, if Bluetooth 4.0 is selected as the preferred communication protocol, all devices that do not support Bluetooth 4.0 are excluded from the network. Furthermore, Bluetooth 4.0 is optimized for distances under 10 meters which means that communication with devices that are further away is not possible, even when both would also support WiFi direct which would be suitable for larger distances.

The last option is more complex and more “futuristic” than the previous ones. It says that devices may use “supporting channels” to detect other devices around them. One possibility is using cameras and a device recognition software to identify devices in the neighborhood. Information about devices and their communication channels may be stored in a (public) database that can be contacted by the initiator device. After the device knows which communication channels can be used, it can directly contact it by using one of the channels. Drawbacks of this method is the large implementation effort since device recognition software has to be implemented as well as a database that holds device information. Second, the method would only work if devices are in sight distance and a camera can capture it.

The considerations above can be summarized by the following question:

**Question:** How can it be ensured that multiple autonomous devices in the local environment can start a communication process without the need of human interaction?
2.2 Data Management

Sharing and linking data was one of the main aspects when the team around Tim Berners-Lee developed the Hypertext Transfer Protocol (HTTP) and the Hypertext Markup Language (HTML). In [BERb] Berners-Lee says that the technology to build up a global network between computer was already available in the 1960’s, however, two essential points were not satisfied at that time: A critical mass of people that are using computers and a stable network where multiple entities can connect to each other. When those requirements were fulfilled, the urge to share knowledge virtually than exchanging papers became stronger. One of the most important aspects of the Internet was its design as an affiliation of independent networks where “small packets of information could be sent from one end of the world to the other with little prior arrangements and a delay of a few tenths of a second” [BERb]. The design of keeping packets small but deliver them fast was a core solution to make connections in the Internet reliable even when the underlying network is not which made the development of the Internet possible in the first place. Furthermore, the idea of linking content through hyperlinks expedited the creation of pools of connected data chunks. Especially this point was critical due to the fact that even in CERN where Berners-Lee developed the first protocols and languages that formed the world wide web, there was no coherent way of structuring data of the different research groups. That’s where the development of the HTML and HTTP came into play which had the goal to unify data access. This process can be split up in three essential steps. The starting point is a “raw” chunk of information as depicted in figure 2.2. Without further information, this (primitive) data chunk can have multiple meanings. By adding a data structure, the plain value can be enriched which could make interpretation (by humans or machines) possible. The simplest conceivable (abstract) data structure is a set. The meta-information a set gives to the observer is, that elements within the same set are interrelated, but without further information, no statement about the type of the relation can be made. This can be done by assigning a key to a value which is called a map. A map is a 2-tuple \( m := (k, v) \) where \( k \) is a key that identifies a value \( v \). A set of
multiple maps is called a key-value-store whereby in most use cases keys are unique although solutions exists where keys are not used to identify the resource uniquely rather than assign a special meaning to it. For example, in HTML, values are enriched by using "tags" which can be seen as keys for assigning further value to a text.

While a data structure makes raw information processable by computers, a data identifier adds a meaning to this data chunk. This two terms are thematically related since some data structures (e.g. a map) may act as data identifiers as well. An important distinction is that keys used in a data structure are always related to a specific (local) application or domain while a data identifier can also act as a global cross-application identifier. Berners-Lee introduced the "Universal Resource Identifier" (URI; later renamed Uniform Resource Identifier) in 1994 to make resources identifiable in the world wide web. An important question in this context is what should be identified. According to the RFC 3986 that holds an extensive description about URIs, the identifier is used to tag "resources" whereby the "specification does not limit the scope of what might be a resource; rather, the term "resource" is used in a general sense for whatever might be identified by a URI" [RFC5]. This means, URIs can be used to identify servers, documents on these servers, services or applications. URIs can also be telephone numbers, mail addresses or names. An URI can further be divided into "Uniform Resource Locators" (URLs) or "Uniform Resource Names" (URNs). URLs - as the name suggests - are used to identify resource based on a location while URNs are used to identify a resource by its name independently of its location. It is noteworthy to say that URIs (and therefore also URLs and URNs) are not necessarily globally unique. This means that different resources can have the same URI but are completely independent from each other. A simple example is given in RFC 3986, namely the URI "http://localhost". The "localhost" identifier is among the reserved Top Level DNS Names specified by RFC 2606 and by the scheme "http", the query is sent to the locally installed webserver. Since "localhost" is bound to the local machine, the interpretation of the URI is not global but must be done locally. However, if URIs should be globally unique, another system has to be introduced and that’s where Domain Name Service (DNS) comes into play. Domain names can be used in URLs and are assigned by the "Internet Corporation for Assigned Names and Numbers" (ICANN). Since domain names are only assigned to one registrant, it is ensured that no one else will use the same domain name in the Internet and therefore all URLs that use the domain name are uniquely pointing to the resource the registrant specifies. Due to the global validity of domain names, this system is commonly used when uniqueness must be ensured, especially for documents or services in the world wide web or namespaces in RDF or XML.

The last part that is necessary to build a data management system is a data access method. A data access method is used to store, retrieve or manipulate data or move them from one source to a target destination. A data access method consists of two essential parts: A language or at
least a grammar that can be used to specific the type of data access and a "Data Access Layer" (DAL) that is used to process the command. It is noteworthy to say that a DAL is not necessarily an interface but can be any type of method or protocol that enables the processing of data access queries.

Two commonly known data access languages are the "Structured Query Language" (SQL) and the "XML Path Language" (XPath). While SQL was invented for database manipulation, XPath has a similar objective but for hierarchically structured data such as XML or JSON. The answer to the question which data access language should be used highly depends on the underlying data model. Table 2.2 gives an overview about three commonly used data models: Table-based, Hierarchy-based and Graph-based. Every data model has its advantages and disadvantages and the assigned languages are specifically designed to fulfill the needs of the particular data model. However, all languages have in common that they usually enable the user to Create, Read, Update or Delete (CRUD) data. A data access language can either be standardized or may be freely defined by the developer.

The second part, the DAL, makes the commands of users processable. A commonly known data access layer is "Open Database Connectivity" (ODBC). ODBC is available for a large number of databases and is a standardized interface for database queries. ODBC can be used with the data access language SQL, this makes it possible to operate with databases of different vendors by using just one language as long as they offer ODBC drivers for their database.

In summary, the following steps are necessary to enrich raw chunks of text and make them usable:

1. Step: Add a data structure. This makes data chunks processable.
2. Step: Add a data identifier. This makes data chunk addressable.
3. Step: Add a data access method. This makes data chunks manipulable.

These steps work fine when the data management system is provided by only one vendor and a requester is aware of the access methods this provider offers. However, problems arise when multiple providers come into account, everyone with different languages, identifiers and access methods. A cooperation between these providers and an agreement on which data structure and data access method should be used would solve the problem but, as stated above, these decisions heavily depend on the underlying data models and due to heterogeneity of providers, it is not a trivial task to find holistic data languages and methods that are still efficient for every use case.

The most promising attempt to solve this issue was made by the introduction of HTTP in the 1990's. Although HTTP is a network protocol in the first place, it can also be seen as both, a data access method and a data access language. In RFC 2616, HTTP/1.1 is defined as an "application-level protocol for distributed, collaborative, hypermedia information systems" [RFCa]. As the two terms "distributed" and "collaborative" suggest, HTTP was designed to make data from various sources accessible through one uniform protocol, namely HTTP. HTTP is also used as a key-access method in the concept of "Linked data". "Linked data" is the attempt to get from "Data Islands to a Global Data Space" [HB11]. It was suggested by Tim Berners-Lee to expedite the "semantic web" and make data of different sources usable. He formulated four rules that should be fulfilled by any participant in the semantic web [LIN]:

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)
4. Include links to other URIs. so that they can discover more things.

URIs and HTTP as data identifiers and data access methods respectively were already used in the early days of the Internet. New is the usage of the "Resource Description Framework" (RDF) along with HTML as data structures. Generally spoken, RDF is a collection of multiple specifications including a data model, a vocabulary (syntax) and a query language (SPARQL). In RDF, a resource "is represented as a network of triples. The three parts of each triple are called its subject, predicate, and object" [HB11]. This structure follows the idea of natural human language as it defines WHO (subject) DOES WHAT (predicate) WITH WHAT (object). There are multiple RDF serialization formats, the simplest one is "N-Triples" where every statement ends with a dot and every triple is separated by spaces. Multiple statements can be grouped within...
The following is a simple example of an RDF graph serialized as N-Triples:

```plaintext
@prefix : <http://example.org#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
:Alice :contacts :Bob.
:Alice rdf:type :Human.
```

The first two lines define prefixes which can be used to shorten statements. The third line makes a statement about Alice, contacts and Bob. All three parts belong to the ":" prefix which is assigned to http://example.org. The complete statement would be <http://example.org#Alice> <http://example.org#contacts> <http://example.org#Bob>. Since URIs are domain names in this case, it is ensured that the resource <http://example.org#Alice> is globally unique and only the domain holder of example.org can change it.

In the last two lines in the given example, a standard identifier rdf is used for the predicate. By using standardized identifiers it becomes easier for others to integrate statements in their own data structure. Figure 2.3 shows an example how such an integration process can look like. Two domains are shown, one (example.org) offers geographical information while the other (inqubu.com) offers information about a company. While some identifiers are just valid in a local
context (e.g.:City or :worksAt), the owl:oneOf identifier is standardized and can be used to ensure cross-domain validity.

The simple RDF scheme for each of the said domains is listed below. By using the query

```
SELECT ?person ?company WHERE {?company owl:oneOf <http://example.org#Austria>. ?person ?x ?company}
```

it is possible to retrieve all companies that are related to the resource "Austria" (as provided by example.org) and all persons that are further related to that company without knowing the predicate of the local RDF of inqubu.com (which would be :worksAt).

RDF is an attempt to solve the so called "semantic integration problem". Semantic integration is the process of heterogeneous data sources with different schemes into one coherent target scheme [DH05]. Semantic integration is a very well studied research area since the advent of computer networking and distributed data sources made solutions in schema integration and conflict resolution necessary.

One semantic integration problem are homonyms. For example, in English as well as in several other languages, the word "bank" has different meanings. Without further information it is impossible to find the correct meaning to the word "bank" (a financial institute or an area along a river). RDF can solve this problem by adding predicates and objects to the subject "bank" as shown in the RDF example above. In the local context of inqubu.com, "Bank" is assigned to the object "Company". Anyone (humans or machines) that can interpret "Company" correctly can assign the correct meaning to the subject "Bank" in this context.
The problem that RDF faces however, lies in the way how data is identified. RDF uses domain names to ensure non-ambiguous identity. This means, data chunks must be stored on a webserver or at least a pointer must be set on a webserver that links to the data set to make RDF work. In other words, all data chunks that are placed on devices to which a domain name is not assigned are excluded from being identified non-ambiguously and this is the case for a lot of mobile devices such as smartphones, smartwatches or smartcars. DOA should address this issue, the question is therefore:

**Question:** Which data structure and data identifiers should DOA use to make data from various sources processable by others even when data is stored at clients without fixed addresses?

### 2.3 Heterogeneity and Fairness

In this section, problems that arise when dealing with heterogeneity should be described. Heterogeneity in context of distributed systems can have many meanings. In this chapter, heterogeneity should be understood as heterogeneity of users. Users try to fulfill different goals with distributed systems: Some use it as storage for data, some use the computing power of large data centers, others perform measurements on it. Dealing with heterogeneity becomes inevitable when multiple entities collaborate with each other. Therefore mechanisms have to be implemented that only allow communication between entities but also specify rules of collaboration after communication has been established. In this context it makes sense to refer to studies about collaboration in human social structures since some principles of cohabitation may be adopted for distributed systems.

Collaboration in a social fabric can be costly. It takes time and resources to contact potential collaboration partners, it may be necessary to negotiate with them since they may want to reach goals that are not the same or even contradictory compared to the own ones and it may also be the case that collaboration partners do not follow defined rules or act malicious. Nevertheless cooperation among humans has a long tradition and appears to be indispensable for a lot of reasons. One of the first “sharing processes” in the history of mankind was the sharing of food. According to [KGHH05], the human species has a special role concerning this topic compared to other organisms. Similarly to other mammals, humans used to share food with strangers even when they do not get anything back immediately. There are several theories why this phenomenon exists:

- **Reciprocal altruism.** This means that resources are given to others with the expectation to get similar resources back in the future (this principle found its way into many languages
as an idiom: "You scratch my back and I’ll scratch yours"). It is commonly explained by the unknown outcome of hunting [KGHH05] since hunting tends to be exhausting and costly in the sense of high calorie consumption. When it fails, a high amount of energy is wasted as well as time. Reciprocal altruism solves this problem as it allows a more efficient allocation of resources. It also allows members of the social fabric to focus on qualities they are good at while others use their skills for things they can do best.

- Producer control. The producer of a resource has several advantages even if he gives the resource away for free. First, he can typically decide how much should be produced, when and where it should be produced. If the producer has a monopolistic position, he may exert prevalence to the market or consumers. Second, a producer can outline the line of approach in the sense of what should be produced. Again, this becomes even stronger when the produced good is rare.

- Nepotism. This means that individuals that are informal related to each other tend to help them to spread their genes (which are similar).

Some of these points may also be relevant for resource sharing in modern networks. In general it can be concluded that the advantages of resource sharing outweigh the disadvantages. Especially knowledge sharing and learning lead to new approaches and innovations that can bring advantages to a large number of people. Furthermore collective solution-seeking may speed up the task of problem-solving which can again bring advantages to a large group. Some of this basic ideas were inherited to the world of computer systems like the Blackboard approach. The idea behind this concept is that a set of computer agents solve a problem together in a similar way like human experts would do it [WS06]. They use a shared knowledge based called "blackboard" where partial solutions to a problem are stored that can be used by other agents to continue working on it. Agents do not have to understand all parts of the blackboard, they can choose chunks they can work with and proceed with them.

This works fine in very narrow limits, namely when the agents, the blackboard and other necessary components are under control of a single provider. If multiple users come into account, some problems and further questions arise, including:

1. How can malicious agents be identified? In an open blackboard system, it is likely that some agents that participate in the knowledge gaining process may act maliciously and manipulate data on the blackboard.

2. Who maintains the blackboard? If the blackboard is stored on a central place, this may be subject to a charge. In this case, a billing system has to be implemented first. If the
blackboard is stored decentralized on clients, they may manipulate data. To prevent this, a manipulation detection system has to be implemented.

3. How can it be ensured that agents can actually communicate with the blackboard? When a standardized interface should be used, the agents have to adapt to the methods the blackboard specifies. However, this makes the blackboard to the center of attention which raises the question if agents shouldn’t be put in the middle.

Some of these problems have already been addressed above, another issue that was not presented yet but is very spread in P2P networks are free-riders. Free-riders take resources but never contribute them to others. According to a study reviewed in [Gup05], 70% of participants in the Gnutella P2P network share no file and 50% of all files are given from the top one percent of all participants. This is problematic for several reasons: First, if only a small percentage of users share resources, the P2P network actually act more like a client-server architecture and all common problems that occur there (bottleneck, higher costs, poor robustness) apply to unfair P2P networks as well. Second, the system performance of the P2P network suffers when only a small percentage share resources [YZLD05]. Third, free-riding is considered to be unfair by other users and can cause that users turn their back on the P2P network. Especially this reason tends to be underestimated when P2P networks are implemented. An experiment that impressively showed the impact of unfair resource sharing was presented [FF03] called "altruistic punishment". The setup of this experiment was a game with two participants A and B. Person A gets a certain amount of money and can decide how much he is willing to share with B. Person B can decide if he accepts the offer. If not, no one will get any money. Even if any amount of money larger than 0 would be a profit for B, the experiment showed that the offer of A will be rejected with a very high probability if the amount of shared money is under 25%. "This shows that responders do not behave to maximize self-interest, because a selfish responder would accept any positive share. In general, the motive indicated for the rejection of positive, yet ‘low’, offers is that responders view them as unfair" [FF03]. The conclusion of the experiment is that users "are willing to punish others at a cost to themselves to prevent unfair outcomes or to sanction unfair behaviour".

To overcome the free-riding problem, several solutions have been proposed in literature. Among them are:

- "Tolerated theft". Users which only take resources but never give them are tolerated within the network and no mechanisms are implemented to stop them. The idea behind this strategy is the assumption that it is more costly for the producer to prevent theft than tolerating it to a certain level. Users that are not willing to pay for resources may find
creative ways to bypass theft protection systems and put a lot of effort to get resources for free. They are probably less wealthy and a controversy with this type of users would not be worth the hassle. Or, as [KGHH05] formulates it: "][A]symmetries between individuals in the marginal value of additional food can lead to contests over packages. The hungry person is more motivated to fight, while the person with more should relinquish some food because the lost food value is not worth the fight".

- Reputation system/economic models. In literature, this is the most commonly chosen way to reduce free-riding. An example is presented in [VCS03] where a value called karma is used to track how much resources a user has consumed and how much he has contributed. These values are stored on nodes called "bank-sets" that are nodes within the P2P network especially dedicated for storing the karma of each peer. This suggestion has the disadvantage that it is not easy to prevent the manipulation of the karma value. Furthermore, a structured P2P network is necessary to find the assigned bank-sets to each node and every node must be identified uniquely, otherwise a reputation based system would not work.

- Enforce sharing. Especially file sharing programs often automatically use the files found on a device without the possibility for the user to turn off this mechanism. The former file sharing network "Napster" for example put all downloaded files into an "upload" directory which made the files visible for any other user in the network [AH00]. However, strategies exist to bypass this enforcement. The first possibility is to throttle the upload bandwidth which makes it unappealing for other users to download files from this device, another possibility is to use firewalls for blocking outgoing data transfer. As it can be seen, mechanisms to enforce sharing that are implemented or controlled at the client side can easily be bypassed and are therefore not sufficient.

The problems described above can be summarized by the following question:

**Question:** On the assumption that nodes in a distributed system try to maximize their own profit and can act selfish or even malicious, what mechanisms should DOA use to ensure a fair allocation of resources in a decentralized network?

### 2.4 Scalability

A system is called scalable "if it can handle the addition of users and resources without suffering a noticeable loss of performance or increase in administrative complexity" [oN94]. Scalability is seen to be one of the core advantages of a P2P architecture compared to a classic client-server
Reliability and administration. In a system with a small number of clients, it is easy to keep track of every participant. With a growing number of peers, it becomes more difficult to monitor the status of the whole network. It is suggested to give as much autonomy to every node as possible and make them independent from other nodes. The reason for this is that in a network with a high number of peers the likelihood that at least some of them are not working gets very high. If the dependencies among these nodes are low, broken peers do not affect other nodes heavily.

Consistency. If a value is only stored at one place, it is easy to manipulate it. If the same value is stored on multiple places (for the purpose of failure resistance), mechanisms have to be implemented to keep the replicas up-to-date. If some transactions on a value happen concurrently and these transactions are contradictory, this can lead to problems. The simplest solution to deal with this issue is implementing locks so that only one write or read access is possible at a time. If it is likely that several instances want to access a resource simultaneously, a simple lock may not scale well. If access to all replicas of a value is possible, strong consistency in an asynchronous system (like the Internet) is not possible, however eventual consistency can be achieved.

Distribution. Large data sets cannot be stored on a single device, therefore they must be distributed among other devices. For scalability reasons, the allocation of data must be done automatically and reliably. If data should be accessed subsequently, a mechanism must be implemented to track, where which chunk of data is located. This tracking mechanisms should not be done on a single server or with a single list since this reduces scalability but must be done within the network. A possible solution is using a structured overlay network like Chord to decentralize the task of finding the right value [SMK+01].

The question concerning scalability can simply be summarized as follow:

Question: How can it be ensured that DOA scales well even for a large number of nodes or a large amount of data?

2.5 Security

Whenever multiple entities are involved in a communication process, many potential threats exist that can affect the correct execution of workflows. For simplicity reasons, in this chapter only
those threats are presented, that are commonly discussed in literature and which are especially
relevant in P2P networks where no global control instance exists.

An abstract concept to classify threats can be found in [JRA14]. According to this pattern, a
threat classification follows 5 basic steps:

1. Threat source: Who is the origin of the threat?
2. Threat agents: This is a class of causers of threats, namely: Humans, environmental and
technological.
3. Threat motivation: The goal of an attacker.
4. Threat intention: The reason for the attack (e.g. intentional or accidental)
5. Threat impacts: Potential effects of the attack on the victim.

The first step alone can be challenging in a P2P environment, especially when peers stay
anonymous and no central instance controls the network. Commonly used identification classi-
fiers (like IP or MAC addresses) can easily be spoofed by the peer and are therefore not reliable
when they are not assigned by a control instance and linked to a real-world identity. This also
leads to problems concerning reputation since without identification it is not possible to deter-
mine which node acted maliciously in the past. Although this issue is critical, there are hardly
any solutions presented in literature yet to solve this. A possible attempt to solve this dilemma
is moving away from the identity-issue and try to solve other parts: For example, the question
arises if it is necessary to implement a identification system when the impact of malicious acts
can be kept as small as possible. If so, there is hardly any need to identify the threat or find out
about his motivation or intention since the costs for the attack are low and can be tolerated by
the system.

Beside the identity problem, open access to a P2P network holds potential for attacks. Actu-
ally, the open-access-problem is a logical consequence of the anonymity of peers since without
identification there is no authorization possible. Finally, if chunks of data within the network are
not encrypted and public available, private data can be stolen easily. If all three characteristics
apply to the network (anonymity of peers $P_A$, open access to the network $P_O$ and unencrypted
data $P_E$) the P2P network should be called fully unrestricted and is defined as $P_A \cap P_O \cap P_E$. Figure 2.4 visualizes the possible restrictions for a P2P network based on the three characteristics. Depending on which of these characteristics should be fulfilled by the network, protection
can differ. For example, if open access to a network is granted and data is unencrypted, security
is ensured by identifying the nodes.

In an fully unrestricted P2P network, the following attacks are possible:
Flooding the network with useless data. When identity of nodes and encryption can not be ensured, everyone in the network can add data anonymously. If a peer has bad intentions he may add large sets of useless data which consumes time for other peers to process or verify and therefore performance in the network decreases.

Infiltrate the network and annex it. If the network is not under the control of a central instance, attackers can implant large amounts of foreign peers that successively take over the workflow in the network.

Spy the network secretly. Without data encryption, attackers could just eavesdrop network traffic. When peers are not identifiable as well, it would be hard to determine who the attacker is.

In summary, DOA must answer the following question:

**Question:** How can possible attacks on the network as described above be prevented or how can the impact be mitigated with the help of DOA?
Data Oriented Architecture

Data-oriented Architecture is a conceptual style for software architectures on middleware layer with the intention to simplify the development of decentralized applications by providing a framework and description of components and their interaction within a software system based on data chunks. Although DOA is located on middleware, in this chapter also references to the application- and network layer are made to provide an overall picture of the system. In this chapter design goals that DOA should fulfill are presented as well as a detailed description of the architecture.

3.1 Design goals

A architectural style typically has a concrete purpose. DOA focuses on decentralized applications and mainly tries to fulfil the properties: Reusability, decentralization, scalability and automatization.

- **Reusability**: This is a core concept in software architecture design and software development. It says that software components should be designed and developed in regard to future use. Reusability makes it easier to adjust software systems even after their finalisation which also increases the flexibility to adopt systems when superior business goals are changed. The data-oriented design paradigm presented in this thesis has a strong focus on reusability as well as SOA, however the implementation of SOA or DOA alone does not necessarily ensure reusability \[F+07\]. The critical part in this context is the decision how components should be designed so that they will be usable for current as well as for future use cases. However, multiple guidelines have been described in literature that should support the decision process. The first principle that is commonly used to achieve reusability
is the "maximization of cohesion and minimization of coupling". Cohesion describes how closely elements are tied within a software component while coupling describes the relation between software components. Another principle is the "divide and conquer" mechanism which is also commonly used in algorithm design. It says that complex tasks should be split into smaller parts which are easier to handle and recomposed afterwards. In SOA, the methods a software component offers are split into multiple services that can be invoked by other methods. In DOA on the contrary, the abstraction level is lower, a software component should not offer services but small chunks of data. Even when this brings more complexity to the calling methods, it ensures a maximum of reusability since operations are performed on data-level. At last, standardized and non-proprietary data-formats as well as data-access-methods should be used. The probability that such formats are continuously supported and actively developed in the future is higher than for not very widespread methods. DOA gives free reign to the developers which data formats can be used but limits the possibilities how these data chunks can be accessed.

- **Decentralization**: DOA is designed for but not limited to a decentralized unstructured P2P network topology. The reason why the "decentralised first" approach was chosen is on the one hand the change of a producer-consumer-environment to a "prosumer"-environment as described in chapter 1. Data is produced on more and more autonomous nodes at the consumer side, therefore it is reasonable to include them as equivalent data sources similar to big web servers or data centers. On the other hand, decentralization could be an important step towards the "democratization of the web". When data is produced extensively on the client side, it should be the decision of the users how their data should be used. Even if no one is forced to publish own data in the web, the advent of social networks in recent years along with the idea of collaborating on and sharing data without great effort needed led to a careless handling of data and privacy [AG06]. Even if users of social networks are basically aware of privacy issues, there are hardly any attempts to change this since companies hand over comfortable tools for communication or organizing in exchange for data of the users. There are two extreme positions to change this: The first one suggests to hide as much data as possible and share just a minimum of it. The other approach is to share basically everything and make everyone share everything but break the link between data chunks and a specific user. When data chunks cannot be assigned to a specific user, they are less useful for data loggers (this of course only works for textual contents and not for media data such as pictures or videos).

- **Scalability**: This is closely related to decentralization. An application based on DOA should work with a theoretically infinite number of underlying decentralized nodes. Scala-
bility is reached by focusing on the nodes in the local neighborhood and share data among them. Multiple of these local networks could connect to other local networks by either using a Internet connection or using hops between these networks as gateways.

- **Mobility**: When devices are not located on fixed places, several issues arise that have impact on almost every layer of computer systems. Starting from challenges in the network (e.g. ensure a persistent connection when a user is moving along multiple base stations) to applications (e.g. data such as GPS coordinates can now be effectively used to enrich user experience). These changes also have influence on software architecture, especially the ability of a software system to deal with dynamically changing resources is needed [SG02]. DOA is based on the assumption that devices are mobile and the devices within transmission range change constantly. Therefore, connections are stateless and conclusive, there is no intention to build up a structured overlay network over multiple nodes since these devices may not be available anymore in the next moment.

- **Automatization**: The most important design goal of DOA is to promote automatization. More and more applications and devices nowadays court for our attention. They inform us about incoming messages, new software updates, low battery power, the low filling level of the toner cartridge in the printer, low disk space, bad connection to the wireless network or missing drivers to install a certain hardware properly. At least some of these tasks could be managed or (partially) solved automatically. Smart mailboxes could classify incoming messages and automatically reply or forward them to the right reference person when the receiver is not responsible for a request. When the printer is running out of ink, a smart device could automatically send a request to a cartridge shop or even order new ink on its own. A more delicate issue is the automatic handling of low disk space when computers autonomously decide which data is outdated or which programs are not used by the users and can therefore be deleted. It would be possible to automatically rank data according to their potential importance to the user and prompt the user when important data should be deleted but delete data with low importance (e.g. log files for certain programs that have never been used by the user) automatically. The vision is that human beings should only set general guidelines and computers than communicate among them and control themselves autonomously without further human interaction. Only in special cases defined before (e.g. sending critical data about the user to others) a permission should be necessary. Concepts to avoid abuse of such automatized systems have been demonstrated even before the rapid growth of interest for artificial intelligence in the 1960's. In 1942, a short story called "Runaround" by Isaac Asmiov was published in which he introduced "three laws of robotics" [AE04]. The story takes place in the year 37.
2015 when two human beings and a robot are sent to the planet Mercury for construction work. In the story, robots are developed with regard to three laws: Not to injure a human, obeys given by humans must be fulfilled except when such an order would hurt the first law and a robot must protect himself except when the first or the second law would be violated. When the humans and the robot called "Speedy" arrive at Mercury, "Speedy" is sent to a selenium mine but doesn’t come back. When the humans send another robot to look after him, they find out that Speedy is running around the selenium pool. They find out, that selenium has a bad influence to the robot, so he tries to protect himself from the harmful effects. But when he leaves the pool, he doesn’t obey to the order the humans give him which would violate law number two. In the end, a human gets to the selenium mine on his own risk and makes Speedy to rescue him. In that case, all could be rescued since the first law is more important for the robot than the last two.

The story illustrates that the problems that arise when computers get more autonomy are indeed manifold (in the story, the conflict between two contrary rules trigger the problem), especially when artificial intelligence is taken into account as well. However, automation in this context should not be seen as a task of artificial intelligence rather than a pool of rules set by human beings. Two examples of successfully automated behavior in computer systems should be given to underline that automatization is already daily routine in computer systems. The first example is routing in computer networks. Routing usually takes place without any great effort made by humans. Routers autonomously discover the lowest cost path to a target and keep their routing table up to date. Once a routing procedure is set up by following simple rules, data exchange can happen without further human contribution.

The second example is the ATLAS particle detector used at CERN for the Large Hadron Collider (LHC). In the LHC, protons are accelerated close to the speed of light and made to collide which produces a cloud of particles that is analyzed. The amount of data that is produced in one second is about 40 million beam crossings whereby it is necessary to add multiple "triggers" (filters) to handle these large masses of data. From the original data, only 0.1% is saved persistently, the rest is deleted by computers within the first nanoseconds after the collision. In fact, the process of finding new particles depends heavily on the correct filtering of the computer systems and since it would be impossible for humans to filter these data sets manually, scientists must rely on the computer systems. Since the presence of large data sets is not limited to scientific context and since the amount of data produced in the world wide web is still increasing, the question arises if users will still be able to handle data manually in the future. This issue also leads to
the question what tasks should remain at the side of the users and which tasks should be completed by computers? It’s also an ethical question if computers should be able to autonomously delete data or refuse access to critical data. As the robot Speedy in Asimov’s story showed, a autonomous intelligent computer system that does not have human characteristics and only relies on a few basic rules is even more vulnerable for decision difficulties than human beings. Nevertheless, the need for automatic data processing will not become smaller in the future.

3.2 Addressees

Since DOA is an architectural style, the primary beneficiaries are software architects and software developers. Nevertheless, the DOA paradigm has also influence on the underlying network infrastructure and is therefore also relevant for network architects and designers. DOA can be either used in private (corporate) or public environments. In the first case, DOA would be used on corporate devices within a company such as smart printers, desktop computers or smart meeting rooms. All objects could then exchange data among each other or forward data to other objects that are out of transmission range. The main difference between a corporate and a public use of DOA are security restrictions since private company data may not leave the company and therefore communication among devices must be restricted. In a public environment however, it is desired that devices exchange data even to strangers. By public use, smart phones, smart car and other devices are meant that are used by consumers and get in contact with other devices at public places such as streets, shopping malls or public buildings. Wherever DOA is used, the purpose is always similar: Give software architects guidelines for the design of applications that make use of the Internet of Things with the intention to expedite communication between heterogeneous devices.

3.3 Architectural style

3.3.1 Classification

Figure 3.1 shows the classification of DOA from a technological perspective. As a middleware layer, it is located between the runtime (typically an operating system) and the applications. DOA is directly accessing system APIs and libraries and acts as a service for application on the operating system. This means, applications can invoke DOA methods and make use of the functionality DOA provides. Nevertheless, application can also have built-in DOA functionality
without the need of additional services - in this case, the middleware layer and the application layer melt into each other.

DOA as a middleware layer has the advantage that the style is platform (and device) independent. DOA can be implemented on every operating system that provides the execution of Turing complete programming languages and makes use of the network layer APIs provided by the operating system. This is necessary since DOA heavily depends on the open access to communication channels. When access to these channels is restricted (e.g. by enforcing user interaction when communication should start) or even completely prohibited, the design goals (e.g. automatization) can not be fulfilled properly.

Figure 3.2 shows a possible simple example how a network based on DOA can be integrated in businesses. Autonomous nodes that implement the DOA paradigm can be treated as service from a business perspective. For that reason one of the nodes acts as a leader that has the job of talking with the back-end service (e.g. a business integration application). The application area for the distributed business network can differ depending on the business cases, in chapter 5 such a real world case is presented.

3.3.2 System model

The following assumptions are made to illustrate how the network and the peers within the network behave. The starting point for these assumptions is a set of devices that are not related to each other and not controlled by a global instance. These devices want to communicate with each other and the following assumptions are made:

• The distributed system is asynchronous. This means there are no boundaries on mes-
sage transmission delays or process execution time and there are no synchronized system clocks. Since it is not possible to tell if a message is just delayed or got lost, it is not possible to ensure both, 100% completeness and accuracy of detecting faulty or idle nodes.

- There are no fixed routing mechanisms, devices build up wireless ad-hoc networks on the fly. The connections are stateless and conclusive.
- New nodes can join or leave the network at any time without preceding notification.
- Nodes in the network are selfish. This means that everyone tries to maximize its own profit and does only collaborate with others when economic incentives exist.
- There is at least one possibility to identify a node within a (local) network (e.g. MAC address of the network card of the device or the IP address of the device). However, it is assumed that these IDs are not global.
- Nodes constantly deliver shareable data (this trivial assumption is necessary since a data-oriented architecture heavily relies on the existence of shareable information).

3.3.3 Design Principles

A system based on DOA follows several main principles that are partially already used in other architectural styles and partially new. They can be grouped in three categories: Data design
principles, node design principles and message exchange design principles:

Node design principles

1. All nodes in the network are autonomous and are treated equally. This is a main principle of P2P networks. Despite the fact that there are no explicit client and server instances, it is still possible that nodes temporarily act as "leaders" to control activities. In this case, universal leader election algorithms (as presented in [San06]) can be used.

2. Nodes in the network are anonymous and only identifiable by their network interface address. This allows high flexibility in networks where nodes constantly come and go as well as high mobility of nodes since no identification mechanisms and no address allocation mechanism (e.g. DHCP) needs to be implemented.

Message exchange design principles

1. Local first, wireless first. This principle states that nodes should first scan their local environment for potential resources they need and if this is not successful, use remote devices (Internet) as a fallback mechanism. Furthermore, devices that implement DOA should have access to wireless channels such as Bluetooth or WiFi direct and try to share data over these channels before they use wired communication. The reason again is that this allows a more flexible network since nodes can easily join the network without the pain of establishing wired connections first. The preferred wireless communication technology for DOA is **Bluetooth**. All devices should make sure that they are able to receive and send data over this channel. There are two main reasons why Bluetooth was chosen: First, it is widely spread worldwide and used by a lot of mobile devices [Blu], second Bluetooth sets a heavier focus on lower power consumption with the introduction of version 4 (Bluetooth Low Energy) compared to competing technologies which makes it more suitable for constant repeating communication processes. However, it is possible to use other wireless technologies in DOA implementations but for the sake of cross-domain communication with entities from different environments and organizations, a communication channel over Bluetooth should be implemented on all devices as well.

2. Share data with every device that also implements DOA. Figure 3.3 outlines the difference between today’s architecture for mobile devices (left) and a potential decentralized future architecture based on DOA (right). Consumers use multiple applications on their devices. If they are provided by different hosts, the databases in the back-end are typically enclosed and not accessible from applications of other vendors. This leads to the result that data
may be stored redundantly from an overall perspective. For example, there exist several applications that constantly collect GPS data from users to predict where traffic jam will occur on different app stores (Google Play Store, iTunes, etc.). The principle behind these applications is always the same: Get GPS tags from the users and aggregate the information on central servers. If users would share those data among others directly and applications would just become viewports on these data chunks, it would be possible to get along without servers and make use of a lot more data than if these data chunks are stored on servers that can only be accessed from dedicated applications. For vendors, the advantage is that they do not have to maintain a database since data is stored decentralized on devices of users (note that hybrid solutions with databases could be used as well). Even if this principle seems critical in terms of security reasons, security risks can be highly reduced by structuring data right. If data such as GPS coordinates stand for themselves and are not bound to a real-world identity (a specific user), data chunks become less valuable for criminals. Privacy is not violated when eavesdroppers cannot determine to whom data belongs and when there are no traces to track back the way data has taken. For this reason, message exchange happens directly between source and target devices without routing mechanisms. However it is possible that data chunks are passed over multiple hops since also data chunks from remote devices are stored on the local device for a limited time and can be passed to other devices. Message exchange follows the "fire and forget" principle, so the communication is stateless. Even though data is not encrypted by default, it would easily be possible to encrypt values when source and target node share the same key. This however would violate the principle of sharing data with everyone but could be a critical business applications. But the conclusion in this scenario can be brought down to one sentence: If data should be kept secret at all costs, DOA is not the right way to go.
3. Data exchange happens automatically without user interaction. In all of today’s platforms that support wireless communication, the user is prompted to allow data exchange over a wireless channel. This is reasonable to avoid security risks but not useful for a DOA based system. First, the repository used by DOA is enclosed and isolated from the rest of the system, so potential malicious code that was brought to the device has hardly any harmful primary effects. Second, even if there is malicious code within the key-value-pair that can potentially be executed by another application, the user has to confirm that he trusts the application which has created the data chunk (see "Data design principles"). Problems occur when users trust malicious applications and grant them rights to execute code, but in this case DOA is not different to any other computer system: If users with system rights want to execute applications (even if they are malicious), computers can’t keep them off from doing it.

Devices that implement DOA constantly scan their environment for devices they can communicate with, listen to incoming messages and send data. This is necessary since nodes can always join or leave the network, there is no stable network the nodes can rely on.

4. Data chunks get deleted automatically after some time. The user should not have influence on which data chunks should be deleted, however every node can decide autonomously how long it wants to store data. To do this, every data chunk contains a timestamp $ca$ that simply stores the date when the data chunk was written to the repository. If $ca.olderThan(t)$ whereby $t$ is a timespan set by a user, the data chunk gets deleted. It is possible to set $t$ to a very short timespan so that is very likely that a data chunk will not be handed over to other devices as well. This can be useful when the receiver can only provide limited disk space or data gets outdated very quickly.

Data design principles

1. Divide large sets of data into single usable chunks of data that stand for themselves and are independent of other chunks, nodes or technologies. This means that the raw data itself should be processable for others, there should not be any dependencies to other data chunks. The payload of a data chunk is a key-value-pair completed by header that holds information on how to process the key-value-pair. If there is a logical relationship between multiple data chunks, they can be combined to a data group, however the single chunks within the group still remain independent. Figure 3.4 shows the scheme used in DOA for structuring the data chunk properties. The "address of the creator" is a globally unique address of the application that has created the data chunk. This requires that all applications that use DOA must define an URI of which unique-
ness can be guaranteed. In practice this can be solved by using domain names. Since only the registrar of a domain has access to it, uniqueness is guaranteed. A sample URL for a company that offers an application that uses DOA can look like the following: `<http://www.company.com/application1#version1>`. With this URL, the version of an application of a company is unambiguously identified.

The "address of authentication service" is an address of a service that can prove that a data chunk was actually created by the application that is stated in the "address of creator" field.

"Timestamp remote" is the time when the data chunk was created at the remote device, "Timestamp local" is the timestamp when the data chunk was brought to the repository of the local device. When the data chunk was newly created, both timestamps are identical. The reason why two timestamps are used is that clocks on the devices in the network are not synchronized, hence "Timestamp remote" can be used to determine if multiple data chunks from the same remote device are newer or older than a specific data chunk (and therefore outdated) while "Timestamp local" is used to delete data chunks after some time. TTL stands for "time to live" counter and is incremented by one when a data chunks was transferred to a new device. Data chunks should discarded from further exchange when TTL passes a certain value to prevent infinite flooding. However, nodes can autonomously decide how high this value should be.

"Checksum" is a hash of the data chunk to prove its authenticity.

"Flag" is a boolean value to indicate if a data chunk should be deleted soon when no newer data chunk from the same creator with the same key (note that two data chunks that have the same key and the same creator are considered to be identical) will be received soon (again, the entity can device autonomously how long it wants to wait until it changes the status of the flag). This flag is necessary to implement a timeout-based garbage collection which will be described further in chapter 5.

Finally, the "payload" contains the key-value-pair(s).

2. Every node follows the subsequent formal structure: Let \( e \) denote an entity that has the ability to store and exchange data with others. This entity consists of a shared and a private repository \( e = (S, P) \) and holds chunks of data that follow the form \( d = (k, v, t, ca, ds, cs, as, of) \) whereby \( k \) is the key (of type string), \( v \) the value (of type string), \( ca \) a (local) timestamp when the data chunk was written in the repository (by the application who initially created it), \( t \) a timestamp when the data chunk was written to the repository by the remote device, \( ds \) a data schema (an URI), \( cs \) a checksum, \( as \) an authentication service address (an URI) and \( of \) a flag to indicate if this chunk is outdated. \( D \) is
a set of multiple data chunks \( D = \{d_1, d_2, ..., d_n\} \). Each data chunk is either part of the shared or private repository \( \forall d \in D : d \in S \lor d \in P \). \( d \) can further be part of a data group \( d_g \) that can hold multiple data chunks \( d_g = (k, D, t, ds, cs, as) \). To avoid redundant information, data chunks that are part of a data group can be stored in the reduced key-value-pair form \( d^r = (k, v) \).

Data chunks are written to the repository by applications via a Data Access Layer (DAL). This layer checks if data chunks that are transmitted to the device are well formed and allows write and read access. Applications installed locally have access to the private repository as well as the shared repository as visualized in figure 3.5. All other entities that are based on DOA have the possibility to only write data to every other entity that is based on DOA. To avoid security risks, the shared repository is sandboxed and the DAL takes care that all data chunks written to the repository follow the scheme given above.

From a developers point of view, the repositories are completely encapsulated from the application. The application writes data into the repositories but doesn’t care about sharing them (the communication and data exchange process is accomplished by the DOA middleware layer). In a practical implementation, a onChange-listener can be registered that informs the application when new data chunks in the repository are available that belong to that application (this can be checked by the address of the creator). In fact, this should take away the pain of wireless connections and data exchange from the developer and bundle data exchange logic in the middleware layer.

3. Data chunks are not identified by the place they are located but by the application that created them. This is a big change of paradigms since today in local networks as well as in the Internet, resources are typically identified by addresses (e.g. URLs). The reason why DOA uses this principle is the higher abstraction level from the network. When
data is bound to a specific address in the network, this node can become a single point of failure, furthermore it has become common usage to store data on multiple hops in large data centers to increase availability and prevent the risk of data loss. The creation of data is driven by applications in DOA, so applications can write any information in the repository they want. There is no central control instances that validates information stored in data chunks. So entities may act malicious and produce data chunks that use the same structure and scheme of other applications. As a result, it wouldn’t be possible for third parties to distinguish if data chunks were really produced by a certain application or by some other malicious entities. This problem is described as the "Byzantine Generals’ Problem" named after the struggle between multiple generals to coordinate a collectively attack with the apprehension that at least one of them sabotages the plans. This problem however leads to the question how it can be proven that a certain data chunk was written by a certain application? To solve this problem, an authentication service is used that must be accessible by the node that want to authenticate a data chunk. The address of this service is stored in every data chunk and can be either a webservice that can be called over an Internet connection or a service that is available within the decentralized network. The authentication service holds a secret private key that is only known by the authentication service and the locally installed application. Whenever the application produces a data
chunk \( d \), the HMAC of the chunk is calculated by using the private key \( k \).

\[
  cs = HMAC(k, d) = H((k \oplus opad) | H(k \oplus ipad) | d)) \quad (3.1)
\]

The result is a hash that is stored in the checksum field of every data chunk. It is recommended to use at least a 128-bit key, however the decision is up to the application provider. When another entity wants to check if the data chunk actually comes from the application as stated in the data chunk header, they call the authentication service that must be provided by every application that uses DOA and pass the data chunk. Since the application and the authentication service share the same private key, the authentication service can calculate the HMAC of the data chunk and compares it to the checksum given in the data chunk header. If these values match, the data chunk is proven to be authentic and a conformation message is sent back to the sender. Notably here is that the data chunk header may be complemented by a version number. This allows to change the shared private key constantly and avoids security risks when the shared private key becomes public due to negligence or criminal activities.

The authentication check described above undoubtedly involves some overhead since the raw data (key-value-pair) is complemented by meta data to give other entities the chance to authenticate the data chunk. Furthermore, authentication services must be provided and hashing consumes time as well. Notably is that the authentication check is optional and not mandatory in DOA if the receiver assumes that a data chunk received is authentic.

The procedure of signing messages with a checksum is already successfully used in digital signatures whereby the authenticity is ensured as well as the integrity of data chunks.

### 3.3.4 Vocabulary

In the design principles above, most of the elements used in DOA were already presented. Table 3.1 summarizes these components and gives a short description to each.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Activity</td>
<td>Runs in background and periodically initiates requests to discover wireless devices in transmission range. The control activity is started when the DOA service is launched.</td>
</tr>
<tr>
<td><strong>Data Access Layer (DAL)</strong></td>
<td>A component that offers methods to read and write data chunks to the public and private repository. The DAL checks if the specified format of a data chunk that should be written to the repository is correct and stores it afterwards, otherwise it gets rejected. Note that the DAL does not offer read access to remote devices, only local applications can read chunks in the repository. The DAL also takes care to increment the TTL counter when a data chunk is written into the public repository as well as calling its authentication service to prove if the chunk was created by the correct application. Furthermore, it deletes data chunks that are outdated.</td>
</tr>
<tr>
<td><strong>Converter</strong></td>
<td>This component serializes data chunk objects to a text string that can be passed over the wireless channel and vice versa.</td>
</tr>
<tr>
<td><strong>Private data repository</strong></td>
<td>A data store that is only accessible through the DAL and only available for locally installed applications. Data in the private data repository is never shared with other users (but data in this repository can be shared with other local applications), hence it can be used for private data.</td>
</tr>
<tr>
<td><strong>Public data repository</strong></td>
<td>Similar to the private repository, but data chunks that are stored here are constantly exchanged with other devices within transmission range. It is noteworthy to say that there is no guideline in which way data should be stored. A database is conceivable just as in-memory stores like B-trees or simple arrays.</td>
</tr>
<tr>
<td><strong>Data Model</strong></td>
<td>A description (a class) that defines how data chunks and data groups should look like. The data model is predefined and must be used by every DOA implementation. It consists of the following fields: Key (string), value (string or array), creator URI (string), authentication service address (string), checksum (string), local timestamp (timestamp), remote timestamp (timestamp), time-to-live counter (integer), outdated-flag (boolean). If &quot;value&quot; is an array, the chunk is treated as a data group (that consists multiple data chunk). If &quot;value&quot; is a string, the chunk is treated as single data chunk. It must be possible for all values to serialize them to strings. The DAL also takes care of garbage collection and deletes data chunks in the public repository which are older than a reference timestamp ( t_d ) and marked as outdated.</td>
</tr>
<tr>
<td><strong>Authentication service</strong></td>
<td>The authentication service is not part of a local DOA implementation but part of the authentication process that takes place when data chunks are written to the public repository. The authentication service receives a data chunk and returns a confirmation if the data chunk is authentic or not. The authentication process is not mandatory, nodes that implement DOA can decide autonomously if they want to check for authenticity.</td>
</tr>
<tr>
<td><strong>Wireless Channel Handler</strong></td>
<td>This component interacts with the wireless hardware module of the device and offers methods to start the discovery process (to discover new devices within transmission range), listening and sending methods.</td>
</tr>
<tr>
<td><strong>Authentication Service Handler</strong></td>
<td>This component takes a data chunk as input variable and calls the related authentication service. Depending on whether the authentication service takes place within the network as local service or via Internet as Webservice, the implementation differs.</td>
</tr>
<tr>
<td><strong>Device List</strong></td>
<td>A list where all devices that are currently within transmission range and use DOA are stored. This list is overwritten consecutively.</td>
</tr>
</tbody>
</table>
Connectors

<table>
<thead>
<tr>
<th>Connectors</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discovery request</td>
<td>Discovery requests are sent continuously from the Control Activity to the Wireless Channel Handler. The Wireless Channel Handler than starts the discovery and listens for inquiry packets of other devices. At the same time, it also sends inquiry packets to be discoverable for other devices.</td>
</tr>
<tr>
<td>Listening request</td>
<td>The listening request is triggered after the discovery request has finished. If the listener detects other devices, it creates an entry for each in the Device List. After a timespan $t_s$ that can be set arbitrarily, the send request is triggered.</td>
</tr>
<tr>
<td>Send request</td>
<td>With this request, data chunks that are present in the public repository are sent to devices that are present in the Device List. Two questions appear here: 1) Which data chunks are exchanged and 2) to whom they are sent. There is no fixed routine since different use cases require different behavior. In the basic implementation, all chunks in the public repository are sent to all receivers in the device list.</td>
</tr>
<tr>
<td>Store chunk request</td>
<td>This event can be triggered either by locally installed applications or after the listening request has finished. It is handled by the DAL.</td>
</tr>
<tr>
<td>Read chunk request</td>
<td>This event can only be triggered by locally installed applications. It is handled by the DAL.</td>
</tr>
<tr>
<td>Delete request</td>
<td>Sent to repositories to delete data chunks that are outdated.</td>
</tr>
<tr>
<td>Convert request</td>
<td>Serializes and deserializes objects to strings and vice versa. This request is triggered by the Wireless Channel Handler.</td>
</tr>
<tr>
<td>Authentication request</td>
<td>Sent from the DAL to the Authentication Service Handler.</td>
</tr>
</tbody>
</table>

Table 3.1: DOA Vocabulary

### 3.3.5 Design rules

Design rules specify how the components of the vocabulary can be connected. Furthermore they define restrictions on these connections.

- The DOA service is controlled by the "Control Activity". It starts and stops the discover-listen-send loop on the wireless channel and offers these methods via an interface that can
be accessed through a GUI.

- The Data Access Layer offers a read/write interface for local applications on the device. It also offers write methods to the public repository for the Wireless Channel Handler.

- The public and private repository are only accessible through the Data Access Layer. The DAL periodically sends delete requests to both repositories to delete data chunks that are outdated. If the outdated-flag of a data chunk is set to false, it will become "true" after this request. If the outdated-flag is already set to "true", the data chunk will get deleted. This timeout mechanism helps to keep data chunks up-to-date but at the same time, it doesn’t guarantee full accuracy (it is not possible to tell if every timeout means, that a node is actually not available or if the message exchange is just slow or incomplete) but full completeness (every node that is not available anymore will be detected by the system sooner or later).

- The DAL creates an instance of an Authentication Service Handler whenever it wants to check the authenticity of a data chunk. The Authentication Service Handler sends an authentication request to the external Authentication Service containing the data chunk as a string and returns the data chunk and a boolean value (authentication successful or not).

- The DAL controls the conversion (serialization/deserialization) of the DataChunk in a way that it can be passed over the wireless channel or is converted from a serialized format back to an object respectively.

### 3.3.6 Analysis

In DOA, the authentication mechanism is of particular importance. As already mentioned, users decide if they trust certain application instead of trusting certain locations (domains). Data chunks that were created by a certain application are accepted, when the user granted permission for this application. In order to ensure that a data chunk was actually created by a certain application and not by a malicious third party that pretends to be a trusted application, an authentication service is used. This authentication service can validate the data chunk and confirms or denies authenticity. Since every chunk must be checked autonomously, the authentication overhead increases linearly with the number of data chunks that are exchanged. The architectural style is therefore not suitable for heavy data exchange, at least when the authenticity of data chunks is important.

Besides authentication, the principle of sharing data with everyone has a deep influence on the architectural model and is also a potential security risk: When data chunks with a ttl counter
of 1 are received, it means that these chunks were created by a device that is currently within transmission range. If there is only one device within transmission range, the attacker knows exactly which device produced these data chunks. That would be a violation of the rule that the identity of devices shouldn’t play any role in DOA and devices should not be identifiable over data chunks they have produced. This means that sensitive data should not be used in DOA rather than data, that has hardly any value for potential attackers but can be aggregated to useful information for users (e.g. GPS tags of multiple users on the street for traffic jam warnings).

3.4 Virtual Organizations

There are multiple reasons why collaboration with others is an advantage compared to solving a certain task on its own. Using resources from others can compensate (temporal) bottlenecks of own resources, shared knowledge can be extended by others (similar to a blackboard system) and a simple trade of resources can bring goods to users that would not have access to it otherwise. There are multiple dimensions on how collaboration can be classified, some of them are:

- Real or virtual: Indicates, if the collaboration takes place in real-world (e.g. a face-to-face meeting in a company) or virtually (e.g. a VoIP meeting over the Internet)

- Flexible or rigid topology: In a mobile environment, nodes come and go. The environment is dynamic and unreliable while in a rigid topology, nodes stay where they are for a long time.

- Coordinated among the participants or coordinated by a central instance: Indicates, if the rules how the collaboration should be processed are predefined by a control instance or participants can negotiate with others about them.

- Driven by a common vision or an accumulation of single interests: If scientists write a scientific paper together, they have the common goal to make it as well-elaborated as possible. By looking at passersby on the streets, it’s harder to find a goal they all have in common but there is a possible starting point to find out: Their physical location! When people share the same location, it is likely that at least some of them have something in common. Some people on the street may look for warnings of traffic jams, some are looking for best prices for a product they want to buy. When they collaborate, they can share information about potential traffic jams on other streets or discounts some passersby just got on a certain product in a shop.

Depending on these dimensions, a collaboration mechanism has to be chosen that is suitable for different situations. DOA is made for collaboration that is **virtual, flexible, coordinated**
among the participants and driven by single interests of the participants. Following the definition of a Virtual Organization in chapter 1.5, every collaboration process that is driven by a common goal (even if its just temporal) is part of a Virtual Organization. Since it is very likely that participants do not only have one common goal, they can belong to multiple Virtual Organizations at the same time. It is also possible that Virtual Organizations can use data that is produced by other Virtual Organizations for their purpose, therefore the VOs are interconnected. The word "virtual" in Virtual Organization should indicate that the formed organization will cross real-world boundaries. Resources are taken from anywhere, no matter where they are located or to whom they belong. For example, a computer that follows the idea of a Virtual Organization could take storage space from a real-world resource A but doesn’t use the memory of A but B instead because B offers a cheaper price for using its memory. To make this concept possible, it is necessary to uncouple the services a device can offer from real-world resources. This is not a new concept since virtualization of computers or cloud computing follows exactly this paradigm. DOA goes one step further and tries to make resources shareable, not only between consumers and cloud-computing providers but also among consumers.

Let’s consider the following simplified example for better understanding that is also depicted in figure 3.6. Multiple users are on the street and carry their mobile device with them. All of these devices have a DOA middleware installed as well as several applications that can access the repositories offered by the DOA platform. User A has just bought a book in a shop called "bookstore" which owns the domain "bookstore.com". The shop has implemented a mechanism that automatically writes the products the user has purchased to the customers shared repository. In this simple case, the data chunks consist of the address of the creator, the key (which is the ISBN number of the book) and the value (which is the price of the book). When the customer leaves the shop and walks down the street, he probably passes by multiple strangers, let one of them call user B. B has currently looked for the same book A has already purchased in a shop called "betterbooks" with the domain name "betterbooks.com". B hasn’t purchased the book but still has the entry in his shared repository. B is looking for the cheapest price of the book in the local neighborhood and has installed an application called "cheapbookfinder" for this reason. Cheapbookfinder makes use of DOA to find cheap books, therefore it is aware of a lot of bookstores and the scheme they use to structure their data. Cheapbookfinder registers a listener on the repository to get informed when a data chunk from the creators "betterbooks.com" or "bookstore.com" with a specific key (the ISBN number of the book) are in the repository.

In DOA, devices permanently scan their neighborhood for potential collaboration partners to get support for a task they want to fulfill. A passes by a lot of people and the majority probably has no interest to find the book A has purchased. But still, data chunks from A are written to
the shared repository of these pedestrians (the data chunks are deleted after short time and are only exchanged with a certain amount of hops to prevent flooding). Now let C be an innocent bystander that is not interested in a book but passes by A and gets data chunks about the book from A. When C passes B, B will get the information about the book that was initially created by A. When this happens, the onChange listener registered by the application cheapbookfinder will inform the application, that potential useful information about the book B is looking for is in the shared repository. Cheapbookfinder then iterates through the list of all books and returns the store with the cheapest price. In a more useful example, the value field does not only contain the price but is a data group that consists of multiple chunks (e.g. price and GPS location of the store).

B doesn’t know that the information about the book was coming from person A, the identity of the user is irrelevant and should not be tracked in any way. Of course, B could just use a conventional price comparison website to find the cheapest book and the process of finding the book would be a lot faster than the process as presented in the example. However, there are also two key advantages: First, in the example, no central server is necessary to find the book. Second, due to the "local first, wireless first" paradigm, the users get information that is bound to the local environment. If B would search the Internet for the cheapest book, he may find a shop that is far away. The concept is therefore convenient for needs that should be fulfilled immediately in the local environment.

A motivation for the shops to support this mechanism is that they can make free advertisement for their products over this platform. When they write information about their products on the devices of the customer, the customer spreads this information to other users when he walks by in the street. It is comparable to a "virtual carry bag" that has not only the purpose to carry goods but is also promotion for the company. It is not likely that this would be abused for spamming since the data chunks are not directly visible for the users but only for the applications.
on the device. When the application the user has installed is reliably, it will ignore data chunks that were created by untrustworthy businesses and only use data chunks created by renowned companies.

3.5 DOA-SOA binding

The book example described above suffers from a substantial drawback: A critical mass of users who use DOA is necessary before the concept would work effectively. In a private context where all devices are controlled by a single instance and endowed with DOA this would not be a problem. However, for a widespread public use, a large number of users would be needed.

To bypass this problem, a concept called DOA-SOA binding is introduced. It means that applications should try to fulfil a task with local resources first and use webservices as fallback methods when a task cannot be solved locally. For this reason, every node that is part of a local network or a Virtual Organization may also be able to contact a remote Broker Webservice. Figure 3.7 shows a simple topology of such a network. A Broker Webservice is in fact nothing else but another participant that uses a DOA implementation and acts as a big store for data chunks. Since the broker is just another participant of the local network with the only difference that it is located somewhere in the web and connected to multiple local networks, no additional logic has to be implemented. The broker uses the same architectural style and design principles like presented above, additionally it may also implement a converter between DOA and SOA so that it is able to send and receive data to classic webservers. The wireless connection handler is supplemented with a handler that can send and receives web-requests. It is conceivable that multiple brokers exist with clearly defined purposes: Some brokers may mediate shopping. The SOA-DOA-binding enables hybrid solutions of central and decentralized networks and may be a solution to unite the advantages of both worlds in one system.
Comparison with Existing Approaches

4.1 Historical development

The idea of sharing data among applications isn’t new. When Apple released its first personal digital assistant based on the "Newton" operating system in 1993, they used a concept called "Soups" to store data. Unlike traditional operating systems like Windows NT where data is stored in files, Newton used a table-based store similar to a database [CAN]. The key idea behind soups is, that data sets are program-independent. This means, data such as contact information can be created and used by phone book application, but it can also be referenced by an email application. Furthermore, "global soups" exist that hold system information like the operating system version or user preferences. This is similar to the concept of a registry which was introduced by Microsoft for their operating system Windows NT 3.1 in 1993 [Rob00]. The "Windows registry" holds central system configuration data as well as application data. It is a hierarchical database that consists of multiple files but can also become a single point of failure when applications accidentally (or consciously) delete or modify important entries. The enormous advantage of the registry compared to other similar methods is a simplified management since in combination with the Active Directory directory service, a lot of clients can be managed with just a few changes. The registry is therefore both: A centralized data store as well as a collection of independent shared data sources.

In literature, the term "Data-Centered Architectures" appears some time. It is mostly used as general description for systems that are driven mainly by data such as blackboard systems. [TUG] says that "One of the most well-known examples of the data-centered architecture, is a database architecture". This means, a data-centered architecture is not a concrete style but a collection of
architectural styles. This further means that DOA can be classified as a data-centered architecture.

The term "Data Oriented Architecture" has been used before in the field of "Data Integration". Data Integration is the process of bringing data from different sources or with different structures into one coherent scheme. In [Jos07], data-oriented architectures are primarily used for developing loosely coupled systems. Components are connected through a middleware infrastructure called "data bus" and can be added and removed independently. It is not further defined what "component" means in this context, it is only stated that a "component uses a data reader to access a data-object on a topic, and a data writer to update a data-object on a topic" [Jos07]. Components can be seen as brokers that interfere references to data-objects but do not transfer the data-object itself. In this context it is noteworthy to say that this architecture style is correctly named "Data-Oriented Integration Architecture". This underlines the fact that the cited paper focuses especially on integration while architectural styles in general can have multiple purposes and integration is just one of them.

4.2 State-of-the-Art

Table 4.1 gives an overview about commonly used architectural styles today that are also frequently mentioned in literature [Sha96]. Each of these styles has a different area of application and is therefore used in different contexts. In general, the main difference between DOA and existing approaches is the comprehensive description starting from the (lower) deployment level to the (more abstract) data level. The disadvantage of this "all-in-one" solution is that the area of applications for DOA is not as extensive as for other styles that are less specific and more general.

DOA comprises different existing approaches on different levels and combines them to a new architectural style. This includes

- a description of how entities are basically connected to each other (deployment level),
- a specification of which protocols are used especially to expedite automation (protocol level),
- a model of how data is created, accessed and shared among entities (data level).

As already mentioned in chapter 2.2, DOA is built upon a similar concept as Hypertext Systems. Both are data-driven and have a clearly defined data structure, a data identifier and a data access method. The main difference is that Hypertext Systems typically use URIs for data identification that refer to a physical address (e.g. a server) while the data location has no
impact in the DOA model. Event though Hypertext Systems are independent of the deployment level and can either be used in a client-server-architecture as well as a P2P architecture, they are primarily used in client-server environments. Hypertext Systems are vulnerable to outages of resources and suffer from fast changes. This problem becomes evident in the world-wide-web and hyperlinks in documents: Whenever the resource that was linked is not available or has moved, the user just sees an error message but will not be redirected to the new location of the resource (except, the webserver explicitly knows the new location and can return a HTTP-301 "Moved permanently" message to the client). DOA avoids linking resources and therefore avoids the risk of producing invalid links, however this also makes traceability of resources more difficult.

Compared to Hypertext Systems, Event-driven Architectures are more convenient for dynamic changes and linked processes. Event-driven Architectures do not focus on data itself rather than the mechanism how data is distributed among entities and which events start this process. In general, an event consists of an event header and an event body whereby the event header gives meta-data about the event such as event id, timestamp or event type while the body describes the event itself [Mic06]. Event-driven Architectures frequently make use of the publish-subscribe pattern for message exchange [Mar06]. The main idea is offering a registration service where independent nodes subscribe for notification for certain events. If such an event happens, messages are sent out to all subscribers. The reaction to such an event can either be a service invocation, a direct change of resources (e.g. create or delete files) or simple informal notification for users without any further action. Similar to Event-driven Architectures, in DOA, applications can register for notifications at the data access layer when data chunks are available, that may be useful for them. DOA frequently scans the repositories for new data chunks that other (remote) applications may have put into it and matches them with the notification-list. For example, if a (local) application has subscribed for notifications when data chunks produced by application A are available and DOA notices that it recently got data chunks from A, the application will get a notification and a reference to these data chunks. In contrast to DOA, for Event-driven Architectures sharing resources is not the main purpose of the architectural style rather than ensuring a reliable service invocation or data manipulation process after a certain event happens. The address of subscribers has to be known and should not change, otherwise a notification would not be successful. In DOA, the supplier of data can stay anonymous for single entities and its address does not have to be known.

A Blackboard System is probably the most divergent architectural style compared to DOA. A Blackboard System has neither a clearly defined data structure nor data identifiers or data access methods. In fact, Blackboard Systems are a very abstract conceptual style that are applicable
for a large variety of scenarios. For example, in parallel input/output operations, blackboards come in handy to optimize the costs for clients when allocating resources [WS99]. But as already discussed in chapter 2.3, Blackboard Systems suffer from several disadvantages that are contrary to the goals that DOA wants to fulfil. For example, the global blackboard can become a single point of failure as it is not distributed among all clients (and if so, this would add several problems to the architectural style) whereas DOA tries to maximize availability of data chunks by distributing them among several other devices. What blackboards and DOA have in common is they both use the idea of data sharing to gain knowledge. Blackboard Systems do so by using the knowledge of experts while DOA uses the data chunks of any producer. This in fact makes Blackboard System more convenient when the goal of the architecture is quick optimization of processes.

As already discussed in chapter 1.3, Service-Oriented Architectures are widely used, but the implementation not always concludes successfully. The main advantage of SOA is loosely coupling between applications. This makes the integration or the change of applications easier, also the extension of existing systems can be performed easier. But modularization comes with the price of more pieces of software that have to be maintained, furthermore the collaboration of multiple services can become tricky when no standardized message exchange formats are used. The main difference between DOA and SOA is their target audience: While all the services in a SOA environment within a certain domain are well known, there is an unlimited number of (foreign) entities that can participate (temporarily) in a DOA network and exchanging data while staying anonymous. This bears risks in terms of network infiltration or network flooding and makes DOA unsuitable for virtual organizations where data should be kept secret at any price. Nevertheless, DOA has the ability to combine the advantages of SOA in a hybrid data-service-oriented environment with multiple use cases as stated in chapter 1.4.
<table>
<thead>
<tr>
<th>Hypertext System</th>
<th>Event-driven Architecture</th>
<th>Blackboard System</th>
<th>Service-Oriented Architecture</th>
<th>Data-Oriented Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main focus of the architecture</td>
<td>Linking data chaining, workflows</td>
<td>Gathering expert knowledge</td>
<td>Services, encapsulation</td>
<td>Data sharing</td>
</tr>
<tr>
<td>Key advantage</td>
<td>High reusability of data, easier data classification and integration</td>
<td>Focus on the events (sensor input), convenient for process chains</td>
<td>Continuous expansion of shared knowledge</td>
<td>Simplify the integration of different autonomous applications by loosely coupling of applications</td>
</tr>
<tr>
<td>Data structure</td>
<td>HTML</td>
<td>Header-Body</td>
<td>Not specified</td>
<td>Mainly JSON, XML</td>
</tr>
<tr>
<td>Data access method/interface</td>
<td>HTTP</td>
<td>Event listener</td>
<td>Not specified</td>
<td>Mainly REST, SOAP</td>
</tr>
<tr>
<td>Reusability of Data</td>
<td>High</td>
<td>Medium</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>Coupling</td>
<td>Tightly</td>
<td>Loosely</td>
<td>Loosely</td>
<td>Loosely</td>
</tr>
<tr>
<td>Robustness against failures</td>
<td>Medium</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Scalability in terms of participants</td>
<td>High</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Primary security flaws</td>
<td>Unauthorized changes</td>
<td>Code injection</td>
<td>Denial-of-service, user authorization</td>
<td>Denial-of-service, code injection</td>
</tr>
<tr>
<td>Support for mobility</td>
<td>High</td>
<td>Medium</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Usable in a P2P environment</td>
<td>Barely</td>
<td>Given</td>
<td>Given</td>
<td>Barely</td>
</tr>
<tr>
<td>Ease of integrability in business context</td>
<td>Medium</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
</tbody>
</table>

**Table 4.1:** Comparison of DOA with other approaches
CHAPTER 5

Technical Realisation Approaches

5.1 Implementation

For testing and simulation purpose, a DOA based application was implemented on Android 5.0 (API 21). It is noteworthy to say that DOA is not restricted to consumer devices or highly sophisticated platforms but can also be used in embedded systems. In the following, the practical implementation is explained in more detail. In particular, the technology for wireless communications, the exchange protocol and the failure detection mechanisms are explained.

5.1.1 Wireless Communication Technology

When Tim Berners-Lee suggested the world-wide-web as a mechanism to structure and manage information, a lot of key technologies such as TCP, DNS or hypertext documents were already invented. The challenge however was to get people to use all the same technologies, otherwise a world-wide communication with different participants would not have been possible. Berners-Lee himself said that "the difficult bit was persuading people to join in. And getting them to agree to all use the same sort of HTTP, and URLs, and HTML. I’m still doing that sort of thing. The World Wide Web Consortium (W3C) is like a club of people and companies who feel the Web is important, and keeping it working is important, and making it even better and even more powerful is important" [BERa]. Berners-Lee already had working webservers and browsers available within the scientific community while the industry showed hardly any interest in this new technology. When they realized the (monetary) potential behind the www, they just had to adopt solutions like HTTP, HTML and URLs for their own purpose and joined the network that was already available.
In the Internet of Things (IoT), the conditions are different. Here, the industry showed interest in this new technology very early, a lot of competitive concepts exist that are not necessarily compatible with each other. Especially the choice which wireless communication technology should be used for machine-to-machine (M2M) communication is still not solved globally, a lot of isolated solutions exist. The problem becomes even more relevant in the light that still new possible solutions are suggested. In March 2015, the "LoRa Alliance" presented a concept called LPWAN (Low-Power Wide-Area Network) [SWI]. LPWAN could be used for IoT applications for wide-area use (e.g. "smart city" applications) based on the cellular network.

If automatized M2M communication should become possible, the entities must agree on which communication channel should be used. The trivial solution is to specify a technology that should be used and ensure that everyone who wants to participate must implement it. As already stated in chapter 3, the way to go for DOA is Bluetooth, for several reasons. Bluetooth is already widespread and used by a lot of devices worldwide [Blu]. Furthermore Bluetooth is still extended and convenient for mobile use (BT Low Energy). However, there are several questions that arise when DOA should be used over Bluetooth. The first question is which communication protocol should be used. For low energy application, the Generic Attribute Profile (GATT) is recommended. It uses simple key-value-stores and is design for the continuous exchange of smaller data chunks. The problem is, that DOA requires additional header information for the key-value-payload which can’t be added in the GATT profile. RFCOMM instead allows more flexibility and a reliable data exchange between the devices. RFCOMM can use up to 30 channels for communication. Compared to other protocols that follow a similar purpose such as TCP (with 65535 ports available) this seems to be low, however Bluetooth use another possibility to assign communication gateways to applications running on a device by the Service Discovery Protocol (SDP). The SDP looks up on applications that use Bluetooth and offer services and returns a list of these services to the sender. To identify a service, a 128-bit ID called Universally Unique Identifiers (UUID) is used. The UUID is not necessarily unique since no global UUID name management system exists. This means, if two different applications use the same UUID and are both executing Bluetooth communication, it will lead to collisions.

In the sample implementation on Android, multiple options were tested. The first step is to register a listener that returns all Bluetooth devices that are found in the local neighborhood as shown abbreviated in code example [5.1]. With fetchUuidsWithSdp the offered services of the remote device can be fetched.
Listing 5.1: BroadcastReceiver and UUIDs

```java
mReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {
            BluetoothDevice device =
                intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            device.fetchUuidsWithSdp();
        }
    }
}
```

When an UUID is used to identify a service, it must be predefined and the listener has to be bound to this UUID as shown in example 5.2. The sender also has to know the UUID the receiver has chosen and creates a socket over that UUID.

Listing 5.2: Sending/Receiving with UUID

```java
UUID uuid = UUID.fromString("fa87c0d0-afac-11de-8a39-0800200c9a66");

// Listening
final BluetoothServerSocket bluetoothServer =
    mBluetoothAdapter.listenUsingInsecureRfcommWithServiceRecord("BTServer", uuid);

// Sending
bs = device.createInsecureRfcommSocketToServiceRecord(uuid);
```

If the service should be used without UUID, no public available functions exist on Android, but hidden methods can be called by using reflection. In this case, a channel number (1-30) has to be chosen and a listener has to be bound to that channel (if the channel is already in use, an error message will be returned). Again, also the sender has to know on which channel the receiver is listening and opens a socket on the channel by using `createRfcommSocket`.

Listing 5.3: Sending/Receiving with channel ID

```java
// Listening
Method m = mBluetoothAdapter.getClass().getMethod("listenUsingInsecureRfcommOn", new Class[] { int.class });
final BluetoothServerSocket bluetoothServer = (BluetoothServerSocket) m.invoke(mBluetoothAdapter, 10);

// Sending
```
Method reflectMethod = device.getClass().getMethod("createRfcommSocket", new Class[] { int.class });
bs = (BluetoothSocket) reflectMethod.invoke(device, Integer.valueOf(10));

For the sample implementation of DOA, the method with UUIDs was chosen since it appeared to be more stable. However, the solution is unsatisfying because the UUID has to be known by every device that wants to communicate by using DOA in advance. Of course it would be possible to predefine a UUID that must be used by every device that implements DOA, however this standard UUID should be standardized by the "Bluetooth Special Interest Group" so that it will be considered by developers in future implementations. A possible "hack" to overcome the problem of establishing a quick and simple connection could be using the Bluetooth device name (the user-friendly name). This name is constantly broadcasted by Bluetooth devices, remote devices can read it by using BluetoothDevice.getName(). Devices that want to broadcast data to remote devices could constantly change their device name by using the method BluetoothAdapter.setName(). The remote devices then use this changing device name for communication. This however is again not satisfying for two reasons: First the device name field is not meant to transmit data rather than being used as identification of Bluetooth devices for humans. Second, the device name field is limited to 248 bytes with UTF-8 encoding. Transmitting data over the device name would be unreliable and too slow for real world applications.

A profile/protocol that fulfils the following requirements would be needed to efficiently implement DOA:

- Keep the overhead for message exchange as small as possible and make the communication process as fast as possible
- Allow the sending and receiving (as well as broadcasting) of messages without pairing the devices and without pre-knowledge about the other device or service (no UUID)
- Allow the free choice of the data structure that is sent over the wireless channel (not only key-value-stores)

Unfortunately, there is no protocol that fully satisfies all requirements. Therefore, two options exist: Either a completely new Bluetooth profile is created that is more flexible and meets the requirements or DOA is built upon an existing protocol with reservations. For this thesis, the last option was chosen.
For testing purpose, the DOA asset management application was also implemented upon Wifi-direct. In Android, the WifiP2pManager class (which is fully available since API version 14) manages the discovery of other peers and the connection to them. An advantage of Wifi-direct compared to Bluetooth is the more automatized finding of new peers. A call of WifiP2pManager.discoverPeers starts the discovery process and a registered BroadcastReceiver can find new devices in the onReceive callback by checking WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION. In this scenario, the operating system takes care of finding a "group owner" (the leader) that acts as the server in this connection itself, so no additional steps have to be performed by the developer. The problem that appeared here was that the devices did not always choose a leader reliably. In a lot of cases, a already existing group owner was not found by newly added devices which compounded the communication process and lead to unreliable results (e.g. discarded messages).

5.1.2 Transmission Protocol

RFCOMM was used in the sample implementation for message exchange. On top of this protocol, the DOA-related message exchange protocol is implemented. Thankfully, due to the design of the data chunks as presented in chapter 3.3.3, no additional transmission protocol has to be specified, the data chunks act as protocol and message payload all in one! The data chunk header already consists of all necessary information that is used for data transmission, especially a checksum, a TTL and a flag to mark outdated chunks. This header is not removed when data chunks are stored in the repositories which means that there is some overhead compared to the payload. This comes in handy however when data chunks are transmitted over multiple hops and still contain the timestamp or the address of the creator of the initial node. With this design, the stateless character of a data transmission process is abolished since the "history" of the data chunk is saved within the data chunk itself. This eases the processing of the chunk in DOA. The data structure that is used when data is transmitted over the wireless channel should be a platform independent non-proprietary data format. JSON is a suitable candidate since it is lightweight and easy to use. The following key identifiers are predefined in DOA: create (the address of the creator of the data chunk), auth (the address of the authentication service), timer (the timestamp when the data chunk was created the first time), timel (the timestamp when the data chunk was created at the remote device), check (the checksum), ttl (the time-to-live counter), flag (the flag to check if a data chunk is outdated), key (the identifier of the data chunk), value (the value of the data chunk). Timestamps must follow the format dd/MM/YYYY hh:mm:ss whereby MM is the month (01 to 12) and mm are minutes (from 00 to 59). Optionally, a time zone value can be added at the end following the format "+00:00". If the time zone is missing, the local time
zone is used. All values in DOA are strings except the "value" field which can either be an array of data chunks or a string. A sample serialization of a data chunk (which can transferred as is over the wireless channel) looks like presented in listing 5.4. This data chunk was created by the application that has the address <http://www.inqubu.com/doa_example> and it uses a web-service for authentication which can be found at <http://www.inqubu.com/doa_auth>. The data chunk was created at 30/04/2015 16:52:13 the first time and it was written to the current device at 30/04/2015 16:52:20. This current device is the first device that received the data chunk after it was created (because ttl is 1) and it is not outdated yet (flag is false).

Listing 5.4: Sample data chunk in JSON

```json
{
    "create": "http://www.inqubu.com/doa_example",
    "auth": "http://www.inqubu.com/doa_auth",
    "timer": "30/04/2015 16:52:13",
    "timel": "30/04/2015 16:52:20",
    "check": "84zrbn2bo2n88732eonc",
    "flag": "false",
    "ttl": "1",
    "key": "273143961963",
    "value": "12USD"
}
```

5.1.3 Application Structure and Deletion Mechanism

The class diagram of the sample implementation is shown in figure 5.1. When DOA is implemented within an app and not as an external service, "Control Activity" offers an interface for the GUI (inherits from the Activity class) and controls the "Wireless Channel Handler". This class offers all methods that are necessary to discover new devices, listen for incoming messages and send data. Note that sending and listening simultaneously is usually not possible on a Bluetooth channel, if listening, discover and send activities would be performed at the same time, this would overload the channel. For this reason, the communication mechanism follows a certain scheme: The device is listening for incoming messages on a second thread while on the main thread the discovery process is performed. If the device is discovering new peers, the isListening variable is set to false and the device stops listening until the discovery process has finished, than the isListening variable is set to "true" again and the device continues listening for incoming messages. In the sample implementation, the discovery process is repeated every 60 seconds. When new devices are discovered, they are stored in a deviceList. In another thread that is looped, the data chunks that are in the public data chunks list are sent to every device that is in the deviceList. DataChunk inherit from the abstract DataModel as well as DataGroup.
Both classes are identical with exception that a DataGroup contains other DataChunk objects in the list value. A DataChunk or DataGroup can be serialized into JSON format by using a converter. In the implementation Google gson was used for this purpose. After the serialization the sendData method of the "Wireless Channel Handler" is used to send the data. When a node receives a data chunk (or a data group), it deserializes it and checks if the format is valid. If so, it (optionally) calls the "Authentication Service Handler" which calls the authentication service as stated in the data chunk header. If the data chunk or data group is valid, it is added to the public repository by calling addValue.

In a system like DOA, special difficulties arise when it comes to failure detection. The term "failure detection" is often used in distributed systems to describe that one or multiple nodes have stopped working correctly. In a environment where DOA is used, the total number of nodes in transmission range is unknown, furthermore nodes can constantly join or leave the environment. For that reasons we assume that a data chunk is outdated, when no new data chunk of the same creator with the same key is received within a given amount of time. Since the local timestamp when the data chunk was created at the remote device is stored in the data chunk, it is possible to tell if a data chunk is newer or older even when the clocks among the nodes are not synchronized (of course in this scenario we assume that the local clocks are not manipulated).

Consider the following example to underline the concept: There are three nodes (A, B, C), the ttl limit is set to 2 at every node (this means the data chunk is dropped when the ttl is greater than 2). Every node can be identified by its MAC address, in this example the nodes use it as a key.

![Figure 5.1: DOA Class Diagram](image_url)
for their key-value pair. The nodes want to get an overview about the network by using DOA, therefore they constantly exchange data chunks and inform others about their current view on the network. The timeout value is set to 1 minute. This means, if no new update of a node in the repository is received, the flag is set to "true". All data chunks that are marked as "true" are deleted in each round. The creator address is the same for every data chunk, therefore it is ignored in this example as well as the checksum and the authentication service.

At time 11:23, all nodes add themselves to the local public repository (node that the clock at device C is not synchronized, it is 1 minute late and writes 11:24 to its repository). Timel and timer are equal since the entry was newly created. Two minutes after start, A and C get into transmission range and exchange data. A add the data chunk it got from C and changes timel to its local time (which is 11:25). Timer is the local time of C, it remains unchanged and the ttl is incremented by 1 since the data chunk was passed over one hop. After 1 minute, C and B get into transmission range. The local time at C is now 11:27, the local time at B is now 11:26. B adds all data chunks it got from C to its repository and adds the current local timestamp to timel, the timer timestamp again remains unchanged and the ttl is added. Now B and C have a full overview over the network. Figure 5.2 depicts the process so far 3 minutes after start (note, that the timel and timer value are constantly updated after every minute for the own data chunk. This means, 3 minutes after start at device A, the own data chunk with key 67-0C-8C is set to the current local time which is 11:26. The same procedure takes place at the other nodes).

Figure 5.2: DOA Garbage Collection Part 1

Consider that A and C get into transmission range again 3 minutes after start and sends all data chunks. C now compares the key and the timer value for each entry and sees, that the entry
with key 67-0C-8C has timer 11:26 while the timer value of the entry with this key has a timer value of 11:23. Therefore, the entry gets updated. The entry with key A7-09-D7 that C receives has the timer value of 11:24 but C already has an entry with the same key with a timer value of 11:27. This means that C already has the newest entry with this key and therefore doesn’t update the entry. The same procedure takes place at node A. The entry with the key A7-09-D7 gets updated (since 11:24 < 11:27), the entry with key 67-0C-8C remains unchanged.

Now consider that nodes B and C are still in transmission range but node A leaves the network. After 1 minute (it is 11:28 local time at node C now), C notices that it hasn’t received an update from A since 1 minute and therefore sets its flag to “true”. If C would delete the entry about A right away, it could get problems when B hasn’t noticed the absence of A yet and sends its chunks to C. In that case, C would notice that the entry about A is missing and creates it. This would lead to an oscillating deletion and creation of the missing node A which can be prevented when C marks an entry as old and doesn’t allow an update on that node when it doesn’t come directly from A (if A would rejoin the network and sends data to C, C would notice that A is the initiating source since the ttl counter is 0 send sets the flag to false again). Figure 5.3 summarizes the process and shows the final state of the network 2 minutes after A has left.
5.2 Use Case Hilti

A proof of concept for possible industrial usage of DOA was created for the Hilti Corporation in spring 2015. The following chapter was created in collaboration with Hilti and gives an overview about the company itself as well as a summary about the challenges the company faces in terms of connected devices and concludes how DOA could be used to simplify this process.

5.2.1 About the company

Hilti was founded in 1941 by the two brothers Martin and Eugen Hilti. The company made its first profit with machine engineering for the German armament industry. After the second World War, the company got into the fastening technology business and released a lot of in-house developments that became very successful on the market. Hilti tools are widely spread on construction sites all over the world today, the company employs 21,000 people in 120 countries whereby about two-thirds of them have direct contact to customers.

In the 1990’s Hilti started to support their business processes by computer systems. An IT department was founded that is spread around three locations all over the world today (Switzerland, USA and Malaysia). To support worldwide collaboration, advanced video conferencing systems and robust data centers are necessary on the infrastructure side. On the software side, customized ERP solutions, cloud computing, file sharing servers and mobile apps are used to support both, end users and internal employees. The goal is to guarantee flexibility for agile reactions on business needs as well as reliability which is ensured by in-house competence and in-house developments. Compared with the past, the position of IT at Hilti has shifted from a purely supporting department to an influential strategic element within the company. The IT vision for the future is still increasing customer satisfaction by standardize IT processes, high availability of IT systems and optimized global applications.

5.2.2 Asset management on the construction site

On large construction sites, usually a lot of people are involved off and on field. For the construction of the Burj Khalifa, currently the highest building in the world with a height of 828 meters, between 10,000 and 12,000 construction workers were on the construction site at peak times [BUR]. Similar numbers are given for the "One World Trade Center“. Around 10,000 workers were included to build the 543 meter tall tower in Manhattan (NYC) which was finished in 2014. Even smaller projects that do not reach these numbers require highly coordinated planning efforts. For the construction of new buildings, a lot of different experts with different tools are working together. The tools that are needed for the workflow are provided by the
construction company which has interest that all devices are used efficiently and carefully since these tools usually are expensive. This leads to the conclusion that tools should be a) maintained regularly to ensure smooth functioning and b) monitored regularly to prevent loss or theft. Today, these steps are often carried out manually but the question that arises in this context is: How can computer systems support the procedure of monitoring device status on the construction site and simplify the device tracking for the construction company?

A possible solution is using smart devices that communicate wirelessly and continuously broadcast their status to a base station. The base station than sends the data to a central server where the device status is monitored. This method is promising but has several drawbacks:

- The base station becomes a single point of failure. If it is not available, no data can be sent.
- Not all devices may be in transmission range to the base stations. A real-time monitoring system is not possible when the construction site is large and the solution requires a direct communication between the base station and a tool.
- The potential of exchanging data directly among the tools without a server (base station) is not used. For example, a smart screw could tell the drill that it is already fixed or should be tighten more.

A possibility to solve this issue is setting up an ad-hoc network among the devices and enable the M2M communication not only between tools and the base station but also among tools. Figure 5.4 shows a schematic landscape of a construction site and different devices that are used. All devices use a Bluetooth wireless transmitter as well as a platform that uses a data-oriented architecture. Each device broadcasts its own identity to other devices within transmission range and listens for incoming messages. The mechanism is identical as the example described in chapter 5.1.3. In the end, every node should be aware of all other nodes that are on the construction site. Since messages can also be passed over multiple hop, the tools form an ad-hoc overlay network. When a device notices that it has Internet connection, it sends a list of the nodes in the network (as the node sees it) to a central server. The current status of the device can then be monitored or device information can be used for statistical analysis (e.g. average uptime of a device, unloading time of the battery of a device, etc).

There are multiple advantages of this approach:
Figure 5.4: DOA at the construction site

- The mechanisms perfectly fits for the conditions on the construction site. Workers are constantly moving around with their tool, when they pass other tools, messages can be exchanged quickly without any human interaction.

- The mechanism is stable since nodes can easily join or leave the network at any time. Therefore hardly any installation effort is needed in advance.

- At the construction site, persistent Internet connection can not be ensured at all time. The presented mechanism allows a hybrid solution of a P2P network and a client-server architecture: The network at the construction site is decentralized and can flexibly update the list of nodes at the server whenever one node notices that it has Internet connection.

Disadvantages of this mechanism are the ongoing wireless transmission of data which leads to heavy battery usage. Furthermore it takes some time after all nodes know about all other nodes in the network, therefore the monitoring is delayed.

Figure 5.5 visualizes the simple setup for the demo. The Real-time Tool Monitor System (RTMS) was simulated by using smartphones. The demo nodes offered a simple GUI with just a single on/off button and a console for data output for debugging. For a real use case, a
display would not be necessary, the whole mechanism can be used as an embedded system and the DOA life-cycle may start automatically when the device is turned on. The payload of each data chunk in the RTMS is the device ID, the battery status, the up-time, the type of the device and the location. Figure 5.7 shows a simplified overview about the main activities in the RTMS implementation. The process starts with an initialization of the necessary adapters (Bluetooth, Wifi) and the setting of parameters (Waiting time for the send-receive-loop, waiting time for send requests to the server, maximum time-to-live counter, a device-list where the MAC address and socket information about available devices is stored, a public and a private ArrayList to store data chunks). After that, the device is set to become visible for other devices and multiple loops are started: A receive-loop for constantly collecting incoming data chunks, a server-loop to send the data chunks that are in the own public repository to the server, a garbage-collection-loop in the Data Access Layer that constantly checks if devices in the device-list are outdated and a send-loop that constantly sends data to available devices in the device list. It is noteworthy to say that in this example implementation, the device is listening for incoming Rfcomm connections by using a UUID (this means this UUID must be known in advance by all devices). If a peer receives a data chunk from a neighbour node, it increments the TTL counter of the data chunk and afterwards checks if the TTL is 1. If so, the node knows that the data chunk was newly created by the sending device and not passed over multiple hops. If this is the case, the oldFlag of the data chunk is set to false since it trivially knows that the node that has just sent the message must be alive. If a device does not receive new data chunks from a device within x minutes (x can be any integer value), the device is removed from the list of available devices within x+2 minutes and all data chunks of this device are deleted (the deletion mechanism was already described in the chapter before). If the TTL is higher than a maximum threshold value that was set in initial state, the data chunk is discarded but the device itself is not yet removed from the device list. If the device recognizes that the "creat", "auth" and "key" information of an incoming data chunk is identical with a data chunk that is currently stored in the repository, the Data Access Layer compares the "timer" (the timestamp when the data chunk was created at the remote device) and checks if the incoming data chunk is newer or older than the existing one. If it is newer, the current data chunk gets replaced by the new data chunk, the old data chunk gets deleted permanently.

The deployment model of the setup is depicted in figure 5.6. The server was based on node.js with express.io extension to enable real-time updates. The server is split in only two files. A server.js file (SocketHandler) and a server.html file (View). The server.js file creates a HTTPS-server by using var server = https.createServer(options, requestListener) and listens on an arbitrarily port (>1023) for incoming connections server.listen(3000). In the sample implementation, no login or security mechanism is implemented so everyone can basically send data to the
Figure 5.5: Real-time Tool Monitor System based on DOA

Figure 5.6: Deployment model
server. If a valid JSON data chunk is received upon a REST request, an `io.socket.emit("rtpush", datachunk)` request is sent to the View (server.html). `Server.html` is listening for incoming push-messages by `var socket = io(IP-address) socket.on("rtpush", function(data) )` and can then display the received data chunks on the webpage. On peer-side, the communication process is invoked by the `WebserverHandler` that is only necessary when peers want to communicate with servers as well and not only among them. Overall, the `Control Activity` takes care of process start and passes the initialization variables to the components.

The source code of the example application was uploaded and published at [https://github.com/christian-com-techniques/DOAService](https://github.com/christian-com-techniques/DOAService)
In this chapter, the questions raised in chapter 2 are answered. Furthermore a simulation was performed to determine the performance of DOA

6.1 Qualitative Analysis of DOA

**Question:** How can it be ensured that multiple autonomous devices in the local environment can start a communication process without the need of human interaction?

**Answer:** A good starting point to answer this question is to take a look how communication works between human beings. The human perception is complex and adapted to the daily requirements, external stimuli are recorded by a lot of different nerves and accumulated by the human brain. What eases the communication process between human beings is the fact that all have similar methods and ranges how communication works. The human ear can hear frequencies from 100 to 20,000 Hz, the human voice starts at 125 Hz and goes up to 10,000 Hz. It is not a big surprise that the frequency range of speaking and hearing matches almost perfectly, it is the basis for vocal communication and humans do not have to agree on which frequency they will speak before they start speaking. What can cause problems however is the "protocol" (= the language) people use. Humans can conclude from a various of things if the opposite speaks the same language: They can simple ask and get an immediate feedback if the other persons understands. They can listen first to determine which language the opponent speaks or they can try to conclude from other variables (e.g. appearance) if the person is from the same region and
probably speaks the same language. They may also choose a language that is broadly understood by a lot of people which increases the likelihood that the opponent will understand immediately.

Applied to M2M communication, things are more complex. While the frequency range people talk with are the same for every human being, there are multiple frequency devices can send on as shown in table 2.1. If there is no agreement between two devices on which frequency they send data, both devices constantly would have to scan their environment for incoming signals on all possible frequencies. This would imply a large overhead for communication. The situation today is that a lot of wireless technologies use a frequency of 2.4GHz for communication since this frequency can be used licence-free in a lot of countries. But even if all devices send on this frequency, they still have to agree on the protocol they use as well. This problem also applies to DOA: All entities that communicate must agree that they use DOA for data exchange before they start the communication process. When this can be ensured, different applications can use different data schemes that perfectly fit for their requirements (this is discussed further in the next answer). However, to make DOA work for consumer centric application (e.g. a traffic jam warning system), a critical mass of users that use this architectural style is necessary. Unless this is not the case, DOA offers hardly any benefit for consumers. For business or scientific usage however, where the nodes are under control of a single supervisor, the problem does not really arise. In all other cases, DOA cannot be used reasonably without a sufficient number of users.

**Question:** Which data structure and data identifiers should DOA use to make data from various sources processable by others even when data is stored at clients without fixed addresses?

**Answer:** How data is stored at the devices is not important, either a persistent database or a memory-store can be used. For data exchange, devices that implement DOA however must follow the scheme as presented in chapter 3.3.3. If this is ensured, own data structures can be defined. The address field (creator) can be used to tell other devices how data chunks that were created by an application should be read. For example, an application uses the URI <http://www.application.com/parkingspotfinder/v1> in their data chunks. On this page, a scheme should be deposited to indicate the structure of the data in the payload. The application may use some defined keywords and a simple comma-separated scheme for its data, developers can then read the scheme on <http://www.application.com/parkingspotfinder/v1> and know that data from this application is split up by commas. Also versioning can be ensured easily, a version number may just be added at the end of the address of the creator.

Fixed addresses of devices are not necessary, only fixed addresses of applications that are used within DOA must be ensured. It is in fact not important where data is placed, it is only important who produced it.
**Question:** On the assumption that nodes in a distributed system try to maximize their own profit and can act selfish or even malicious, what mechanisms should DOA use to ensure a fair allocation of resources in a decentralized network?

**Answer:** This is clearly one of the most challenging topics in DOA, for one reason: Devices do not necessarily have to be identified, it is a core principle of DOA that devices can stay anonymous. For this reason, any method that uses a reward-system for fair resource allocation can not be used. Since data chunks are constantly exchanged between devices and users can use data from other devices without any effort, in a pure DOA implementation, the freeriding-problem can not be solved. However, applications that use DOA only work properly when they share data. A traffic jam warning system would only be useful when the location and speed information of multiple users are exchanged. This means that the application shares and consumes data at the same time. But it is also possible that another application uses the data chunks that were produced by the traffic jam warning application without contributing any data. The question however is, if this problem has a impact on the whole system. If a group of users exists that contributes traffic information, the system will also work even when an even larger group only uses the data but does not produce it. Furthermore the data is spread over multiple devices so there are no costs for computing power or energy since the hardware is provided by the users. A lot of application today use a "freemium" business model where the basic version is free but users have to pay for more advanced features. According to [SWR], 50% of the revenue of mobile games is made by 0.15% of the users, the overwhelming majority doesn’t pay at all for the application. But still, the model works quite well in a lot of cases as [Hei15] states. A similar conclusion can also be drawn for DOA systems: Freeriding can be accepted since it should hardly have any grave impact on the system performance. A practical evidence for this statement is part of future research.

**Question:** How can it be ensured that DOA scales well even for a large number of nodes or a large amount of data?

**Answer:** Since DOA is P2P-based and there is no single point of failure, the general scalability performance is high. The number of data chunks \( m \) that are passed within the network can simply be calculated through

\[
m = \sum_{i=0}^{n} x_i^{n-1}
\]  

(6.1)

where \( n \) is the total number of nodes within transmission range and \( x_i \) is the number of data chunks at node \( i \) under the assumption that all data chunks are exchanged with all other nodes. The equation shows an exponential growth of the number of data chunks exchanged depending
on the number of nodes in the network which is not desirable - a linear growth would ensure stable scalability. There are two reasons why good scalability is still achievable in practice: First, DOA follows the “local first, wireless first” principle. The possibility for nodes to exchange data is limited to the transmission range of the wireless adapter. This is a natural constraint on the number of possible receivers for data chunks. Second, even if there are a lot of nodes in transmission range, it is not necessary that data exchange happens between all of them. To increase efficiency, it is desirable that a node only sends data to those nodes, where the likelihood that they need the data chunk is high. This is of course not a trivial task, a possible approach for this is presented in the last chapter.

**Question:** How can possible attacks on the network as described above be prevented or how can the impact be mitigated with the help of DOA?

**Answer:** In DOA, all nodes are autonomous, there is no control instance to check if nodes follow the rules. Malicious nodes can manipulate header information of data chunks or impair the performance of the network. In the following, a non-taxativ list of possible attacks and comments about them are given:

- **Flooding the network with useless data.** In this scenario, a malicious node fires constantly data chunks to other peers within transmission range. The processing of these useless data chunks takes time and decreases the performance of the receiving nodes. Since nodes are anonymous, there is no way to identify the attacker by an unique ID and block the traffic. However, it is possible to block incoming traffic from certain nodes based on the MAC address, if a node notices a high growth of messages from a certain node. But if the attacker spoofs its MAC address, there is again no way to filter the data chunks. This would be similar to a DoS (Denial-of-Service) attack and can not be fully prevented.

- **Spoof the creator address and/or the authentication service address.** A node may produce data chunks and adds a wrong creator address to the data chunks to make other nodes think, the data chunk was created by a trustworthy application when in fact it was not. In this case, the authentication service can be used as described above. But, if the malicious node also tampers the authentication service address, this mechanism wouldn’t work anymore. For this reason, it is obligatory that a node always checks if the basename of the creator address is identical with the basename of the authentication service. For example, if the basename is <http://www.application.com> and the address of the creator is <http://www.application.com/app/v1> then the authentication service address must start with <http://www.application.com> as well. If this is not the case, the attacker could add
any URI and forwards the authentication address to a malicious side, that always returns
"true" on the question if the data chunk is authentic.

- Manipulate the TTL counter. A malicious node can set the TTL counter to 0 and forward it to another node. The receiver then thinks that the sender was the initial creator of the data chunk when in fact it was not. This attack cannot be prevented but has hardly any impact on the system since the TTL counter is only used to drop data chunks after they have passed a certain amount of nodes. A non-malicious node would still drop the node when the TTL counter reaches the threshold it has set, malicious nodes can only extend the time a data chunk will be passed when they set the TTL back to 0, but this has no grave impact on the system when the number of malicious nodes compared to correctly working nodes is low.

- Eavesdrop the network. This is in fact not an attack since data chunks passed through the network are public available as defined in chapter 3. Since exchanged data should not be secret anyway, it is irrelevant when it is intercepted.

To sum up, there are possible attacks on a DOA network, but most of them do not affect the system performance seriously. Attacks that do have serious influence on the system such as DoS-attacks are not a specific DOA problem, all kind of networks suffer from these types of attacks.

### 6.2 Quantitative Analysis of DOA

The evaluation of an architectural style and the finding of convenient metrics to measure the system performance highly depends on the underlying use case. For DOA, several possible metrics come into question:

- The overall battery usage compared to similar wireless ad hoc networks
- The throughput of data between two nodes
- The total number of messages that have been exchanged among the peers
- The time that passes until a message that was created at one end of the network reaches the other end (Information exchange speed)
- Impact on system performance of malicious nodes in the network (e.g. slower message exchange when x% of nodes in the network act malicious and hold back data chunks)
The time that passes until a node has collected x% of all data chunks that were created at other nodes

The robustness of the network (number of crashes within a given amount of time)

Depending on the focus of the application that was built based on DOA, some metrics may be more important than others. In the following, three metrics were evaluated in more detail, namely the time that passes until a message that was created at one end of the network reaches the other end (since speed may be a crucial factor for a lot of applications that use DOA), the battery usage of a device running DOA with Bluetooth or Wifi-direct compared to a device that runs idle and the overall number of messages that are exchanged in the network.

6.2.1 Information exchange speed

For the measurement setup, 2 to 5 smartphones running Android 4.3 with a DOA implementation as presented in chapter 5 were used. On all devices, Bluetooth is supported and used for the communication process. The devices were placed uniformly in line with a distance of 3 meters between each device, figure 6.1 illustrates this setup. For evaluating DOA, the overall time was measured until all nodes were aware of all other nodes in the network. The measurement was performed with 2, 3, 4 and 5 nodes, the process was repeated 30 times for each setup, figure 6.2 shows the result.

The median of the time it took until all nodes knew each other was 29 seconds (for 2 nodes in the network), 37 seconds (for 3 nodes), 110 seconds (4 nodes) and 108 seconds (5 nodes). It is interesting to observe that the median only changed a little when going from 2 to 3 nodes, but there is a bug gap between 3 and 4 nodes. In the best case, all nodes should have known each other within 15 seconds, because every 15 seconds the list of all nodes that is known by a node was sent to any other node within transmission range. The question is: Why are these numbers far away from 15? One possible explanation is that devices coincidently sent data at the same time. In the implementation, the listening process was interrupted when the sending took place, if both devices are sending simultaneously (or listening both without sending) of course no data exchange would be possible. Since the procedure was repeated 30 times for each setup, it is
however unlikely that this is the only reason. When we take a look at the distribution, we will notice a clear positive skew on the histogram for 2 nodes. For 3 nodes, we still see this positive skew but in a mitigated form. The positive skew vanishes completely for 3 nodes and even for 4 nodes, the distribution was similar to a normal distribution. The standard deviation for 2 nodes and 3 nodes are similar (28.65 and 28.87 respectively) while the standard deviation for 3 nodes is much larger (38.61) and is decreasing again for 4 nodes (22.63).

It was observed, that the Bluetooth data exchange was not always reliable while the measurement took place. Even when all devices were running, some devices had problems to find any other device for minutes. It is possible that the Bluetooth hardware adapter had influence on this, because 4 different device types were used (Samsung Galaxy S2, Samsung Galaxy S4, Samsung Galaxy S5, Google Nexus 5 and LG-D722). Furthermore it is also possible that interference of the waves led to mitigation or noise of other Bluetooth devices in the environment disturbed the connections. It was not expected that the median time of finding all nodes in the network was increasing in this way, since every node acts as an multiplier and sends the list of all nodes it knows to all other nodes in the network. Therefore it would be expected that the network is getting more stable and reliable when multiple nodes are added. The reason for the "2-3-gap" and the reason for the different types of distribution couldn't be clarified conclusively.
6.2.2 Battery usage

To measure the power consumption of DOA, first, the battery usage of a Samsung Galaxy S2 which ran idle was measured. To avoid distortion, the 3G connection and the display were turned off, however, several necessary Android background processes were running. Afterwards, the DOA app was deployed on the same phone and the same setup and and the battery usage was measured again in Wifi-direct mode and Bluetooth mode. The result is depicted in figure 6.3. The X-axis shows the time line, the Y-axis the battery status (starting from 100% to 40%). The diagram clearly shows that the battery usage was the lowest when the DOA application was not running on the phone (and Bluetooth and Wifi were turned off) and it was the highest when DOA was running based on Wifi-direct. The measurement however has only limited significance. While for all three measurements the same phone was used, it cannot be ruled out, that the operating system had executed some background tasks which may have distorted the measurement. It may also be possible that the implementation of the Wifi or Bluetooth standard is different on other devices which could lead to other results. However, the comparison gives a first impression how the power consumption of DOA using different wireless technologies looks like and Wifi-direct has clearly the poorest performance.
6.2.3 Total number of messages

"Total number of messages" means the number of successfully received messages in DOA based on Bluetooth including messages for peer discovery and messages for exchanging information about all nodes in the network. Three different types of devices (Samsung Galaxy S4, Google Nexus 5 and LG G3) were used, a message counter was implemented on each device. The measurement took place for one hour, figure 6.4 shows the result. It can clearly be seen that there is a large downturn of messages between 500 and 100 seconds at two devices. This can be explained by the fact that after around 8 minutes, the system reached a "stable state" since all peers have discovered each other and the network state didn’t change as this would make heavy data exchange necessary. However it is interesting to see that the LG G3 didn’t follow this scheme: Here we can see a more consistent drop from about 15 to 1 message after one hour. There are several possibilities what has caused this difference: Maybe the connection to the LG G3 was unstable and the other devices had to resend their peer discovery request multiple times. Maybe the opposite is the case and the LG G3 uses a very advanced Bluetooth adapter which is more reliable and achieves better results when listening for incoming messages. To clarify this, a messages-sent counter could be added on each device and compare the number of sent and received messages.
6.2.4 Conclusion

The measurements performed were just the first step towards a more precise and convincing survey. There are many variables that could have influence on the reliability of the connection, for example it was observed that the reliability of the connection was decreasing when the battery of the device was low. Vendor specific implementations or background processes or "hidden variables" had influence as well on the performance and may have distorted measurements. Therefore, for reliable performance statements, a lot more measurements would have to be performed, especially also on other devices than smartphones since DOA is not bound to specific hardware.
Conclusion and Future Work

7.1 Improvements

There are several potential improvements for the DOA concept itself as well as the practical implementation of DOA. One of the most urgent questions is how the data sharing process can be improved to become more efficient. In the current implementation, all data chunks in the public repository are sent to every other node within transmission range. But it is very unlikely that the majority of the data chunks would be used by others since ... Therefore, a mechanism has to be implemented that makes it possible to predict if it’s likely or unlikely that the receiver can make use of the data.

As a measure for user similarity, applications are used that are installed on a device. The

![Figure 7.1: Example of two entities and their bloom filter](image-url)

Figure 7.1: Example of two entities and their bloom filter
assumption is that the likelihood, that users can profit of sharing data with each other is higher, when the have similar applications installed. As already mentioned, applications in DOA must have a unique URI which can be used to identify the application. However, since data exchange should happen fast and users may have a lot of applications installed on a device, a loop through the list of all applications when users meet may be too slow. Furthermore, users would have to establish a connection to every other device first to check, if the other device has similar applications installed. A better solution for this scenario is using Bloom filters. Bloom filters are already widely used in different scenarios, some browser use them as a fast method to detect malicious websites. A 128-bit array and the Fowler–Noll–Vo hash function is suggested for this purpose. The URIs of all applications are hashed and the output is stored in the bit array, the array is constantly updated when new applications are installed or deleted. Figure 7.1 shows an example of two entities, some applications installed on them and their bloom filter. The bit array is always visible for others and should be broadcasted constantly. When another entity comes into the transmission range of the first entity, it checks the bit array by calculating the Levenshtein distance between the own array and the foreign array. The Levenshtein distance is ranged between 0 and 128 where 0 means a high similarity and 128 a low similarity. Every entity can set a threshold to decide how high the similarity should be to share data. When the threshold is exceeded, data chunks in the public repository are sent to the foreign device. The disadvantage of this approach is, that devices first have to check the bloom filter before the data exchange can happen - this slows down the process. Furthermore there is no built-in approach in Bluetooth to make this approach work efficiently. It would be possible to "misuse" the UUID of a Bluetooth service as an array where the Bloom filter values are stored. It would also be possible to use the "user friendly name" of the Bluetooth device to store the Bloom filter values. The advantage is that this name is broadcasted per default to every other device, however it is not intended to be used for this purpose. Therefore all these approaches suffer from the fact that Bluetooth doesn’t perfectly fit the needs of DOA - and hardly any other wireless technology that exists today does as well.

### 7.2 Conclusion

In this thesis, an architectural style called "Data-Oriented Architecture" (DOA) was presented. The main principle behind this concept is that autonomous nodes form a wireless network and every application which is installed on a node offers public chunks of data to others with a clearly defined scheme for the sake of reusability. One of the key advantages of DOA is robustness since new nodes can join the network at any time and nodes can leave the network at any time without any notification mechanisms needed and without any pre-configuration. Compared to
a structured overlay P2P network, DOA has no fixed routing path for message exchange which eases the setup and installation of nodes. Furthermore DOA is well suited for decentralized applications where no Internet connection is available. But DOA can also be integrated easily in a service-oriented landscape, therefore the style can also be used for devices that have temporal access to the Internet and try to keep themselves up-to-date by communicating with other nodes when no Internet access is available. DOA also tries to promote the idea of the "democratization of the web" because it treats all participants in a network equally and data is shared among users and does not have to be put on central servers of large companies. In general, central servers are not needed for communication, the devices of consumers are not only passively used for consuming resources but are actively involved and used for data sharing.

DOA is an approach to make the Internet of Things possible. An important aspect is that computers should support human beings in the background with a minimum amount of necessary interactions. To fulfil this, it must be ensured that devices can start a communication with others autonomously and get data they need to solve a task. In this thesis, several suggestions were made to automatize data exchange and ease the development of decentralized applications. However, practice has shown that it is very unlikely that DOA will be widely used in the future for several reasons. First of all, DOA is undermining the business model of big service providers such as Google, Facebook or Yahoo since data of users becomes public available and is therefore a shared good. The principles of DOA are not conform with the interests of these service providers, therefore there is hardly any motivation for them to support this scheme. But DOA only works efficiently in a global view, when a lot of devices support it. Without the help of the industry, a large spreading may not be possible. Second, the usage of P2P systems is slower and often unsatisfactory when people are searching for a certain piece of data compared to a client-server-system. Data chunks need time until they have spread across the network, data chunks that are stored on a server are available instantly on a fixed address. This makes DOA less attractive for applications that are updated very often especially when the network is large.

Even when the architectural style will most probably not be broadly used, this thesis might have provided impetus and inspired others to go a step further towards a "smarter future". It should be the goal of the whole mankind to push forward the interconnection between humans of the world to enable collaboration among them. New ideas are often born in exchange with others, knowledge that is not primarily useful for ones may come in handy for others. Computers should support us in our attempt to make the world a better place and become an "invisible partner" for us. DOA was one try to go further in this direction, I’m sure, it will not be the last.
Bibliography


Rohit Gupta. Protocols for sharing computing resources and dealing with nodes’ selfishness in peer-to-peer networks. 2005. [Gup05]


Hilti connectivity landscape. [Company internal used image]. Accessed: 2015-05-18. [HIL]


Tobias Hürter and Max Rauner. Faszination Kosmos: Planeten, Sterne, schwarze Löcher (German Edition). Spektrum Akademischer Verlag, 2008. [HR08]


Morozov: "daten sollten keine ware, sondern öffentliches gut sein". [http://derstandard.at/2000012859075/Internetkritiker-Morozov-Daten-sollten-keine-Ware-sei


Curriculum Vitae

Personal Data

<table>
<thead>
<tr>
<th>Name</th>
<th>Christian Vorhemus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nationality</td>
<td>Austria</td>
</tr>
</tbody>
</table>

Education

- **2013–2015** Master study Business Informatics, University of Vienna, Austria.
- **2013** Study abroad at the University of Illinois, USA
- **2009–2012** BSc in Computer Science, University of Vienna, Austria

Professional Experience

- **2015** IT Application Development and Master’s Thesis, Hilti AG
- **2013** Internship Technical Evangelist, Microsoft Austria
  Developing Windows 8 applications with Windows Azure integration as well as staying in touch with the developer community
- **2010–2013** Teaching Assistant, University of Vienna
  At the chair of "Future Communication": Scaling experiments in virtual machines, virtualization
- **2009–2010** Employee, Novarion IT Service GmbH
  Programming (C#), design and layout of folders and websites