University of Osnabrück

A report about:

Developing holistic software solutions through integration of existing individual solutions.

Connecting Software
An overview about challenges, solutions and the need for software-based integration tools.

Author
Henrik Gerdes
MatNr: 4242442

Supervisor
Dennis Ziegenhagen
Prof. Elke Pulvermüller

February 25, 2020

Contents

Abbreviations

API	Application Programming Interface
DSL	Domain-specific language
EAI	enterprise application integration
EIP	enterprise integration patterns
ESB	Enterprise Service Bus
FTP	File Transfer Protocol
IFTTT	If This Then That
IoT	Internet of Things
JSON	JavaScript Object Notation
OS	Operating System
POJO	Plain Old Java Object
REST	Representational State Transfer
SFTP	Secure File Transfer Protocol
SOA	Service Oriented Architecture
SSH	Secure Shell
XML	Extensible Markup Language

Connecting Software
An overview about challenges, solutions and the need for software-based integration tools.

Henrik Gerdes

February 25, 2020

Software solutions, especially for enterprise applications, currently demand extensive requirements. To meet these requirements, developers started to split monolithic applications into smaller parts to gain better scalability, maintainability and make them future-proof. This paper presents general principles and examples to connect these smaller applications allowing them to provide the same functionality as monolithic applications. Furthermore, this paper also presents several services for end-users to interconnect individual IoT devices from different manufacturers to work together.

1 Getting started. Why connecting?

1.1 Software development then and now

The Zuse Z3(1941) is a great example of this. It's the first functional and programmable computer ever made. It was mostly used to calculate matrices for airplane wing flow. [Z302] This machine had no connection to other hardware and processes.
Nowadays, requirements and the way society use computers have changed completely. Computers are no longer part of an isolated solution search. They act as several chains in a whole process of problem solving. For some processes human interaction isn't needed at all or minimized to the least. Development of software changed too. For the most part software is not developed for one single computer. With the abstraction layer that modern operating systems (OS) provide, applications get developed for a whole platform of computers. Cross platform frameworks and webservices go even further. [Tan09]

Using many computers instead of one provides much more processing power and could save valuable time and money. The drawback is that this adds a whole bunch of new problems. The applications must provide a way to communicate with each other. Developers have to make sure that information from one application gets send, received and understood by another application. The next section will dive deeper into these and other problems.
As already said, our requirements on application have grown a lot. For an enterprise application it is colossal, near impossible task to keeping all these requirements in mind and implement them in a robust and stable way. An alternative way is to create smaller and more simple applications and connect these.

1.2 The diverted and current state of applications

Since the requirements of application grew, so did their number and complexity. Looking into enterprise applications some monolithic applications can be found. These applications bundle a variety of tasks. Figure 1 shows a retail application which consist of front-end webpage, order management, billing and payment parts. These individual components are tightly integrated and deployed as a single unit. For the most part these components use their one proprietary way to communicate with each other via a point to point system.
For example: The webpage has to check with the order management if the requested item is in stock and its delivery time.

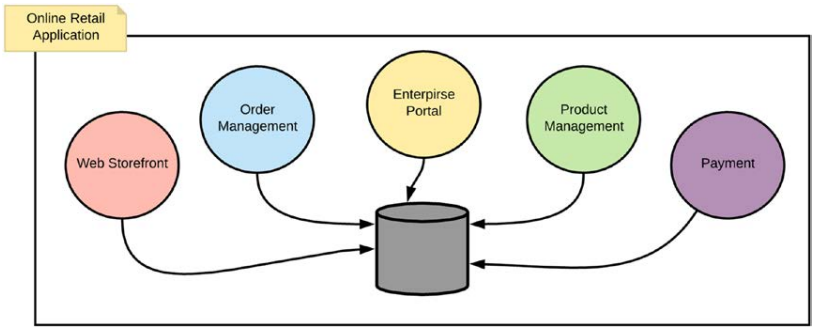

Figure 1: Monolithic retail application
Source: K. Indrasiri P. Siriwardena, Microservices for the Enterprise[IS18]

Updating or replacing the order management requires changes on every other part of the system and full new deployed of the hole system. Application designed like that are hard to maintain, expensive and make it more difficulty to adopt new technology. It also entails the risks that one unstable component can endanger or take down the entire system. [IS18] This is where a Service Oriented Architecture (SOA) and an Enterprise Service Bus (ESB) come in. These paradigms will be explained in section 3.1.1.
Beside monolithic enterprise applications there are thousands of different applications in the end-user space. At the time of Internet of Things (IoT), people experience a high growth of connected devices. The global smart home market is expected to grow to USD 119.26 billion by 2022, an increase of 25%. [YLL18] These devices are designed for several purposes like heater control, energy motoring, security enhancements and common support services. Most of them are manufactured by different companies. To ensure a pleasant user experience these devices should work together. While there are standards out there like Zigbee (IEEE 802.15.4) [K+03] and WLAN (IEEE IEEE-802.11) these standards mostly handle the connection and not the translation of transmitted messages. Besides this not all manufactures use these standards. Therefore the consumer is on his own with a smart speaker that can't turn the light on with light bulbs from a specific manufacturer.
To avoid this a translation and communication layer is needed.

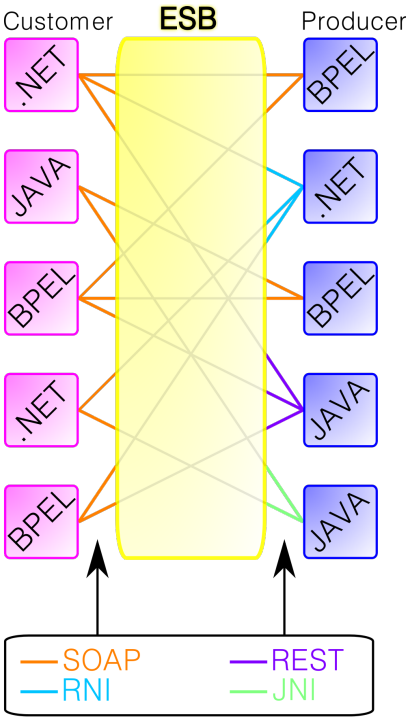

Figure 2: An ESB in a Service Oriented Architecture
Source: Wikimedia Commons[Com18]

2 Problems with integration

The situation described above requires some kind of translation between devices and applications. To solve this problem there are two native solutions. The first is to build a translation application that has no other functionality than receive, store, translate, route and send message from one application to another. Its only purpose is to integrate other application. This approach is often called an Enterprise Service Bus. The second solution is to directly integrate the needed applications with each other. The second solution is analyzed in section 2.1.

2.1 Direct integration

Integrating separated applications with each other gives some advantages. It completely cuts the cost to specify, develop, test and maintain a complete additional application. Depending on the size of the app this reduced outlay can save a significant amount of money. It also reduces the overhead and any potential delay for routing a message through an additional application. Seams great but with these advantages come a bunch of disadvantages. By integrating every application with each other there is an additional cost for every application in the system. Every application might need to communicate with every other application. This causes some challenges in development. The developers of application A need to have some basic understanding on application B or at least on their API. On the implementation side arise problems that application A has to block till it received the acknowledgement that application B got the sent message. Assuming a domain-specific layer and applications based on different frameworks this could cause problems. Frameworks are mostly designed for extension not for integration. [MBF99] As said in section 1.2, the addition of another application or the change of an existing one requires changes on every other application.
Among the problems described above are additional problems that every network application has to face. Networks are slow and unreliable. Realizing integration on an application by application layer means to deal with each of these problems on every service. [HW04]

2.2 Using an Enterprise Service Bus

Analyzing the other option reveals that many of these problems can be avoided by using a separate application for integration. These applications are often called ESB. The concept is to allow communication between multiple applications. It is used to allow interaction between Service Oriented Architecture (SOA) and other multi-application environments. A SOA is an architecture concept which in which applications provide their functionality in the form of reusable services. The service itself is a self-contained application that

represents one business function. A service may consist of multiple other services and hides its internal functionality. [Men07]
An ESB provides the advantages of loose coupling by reducing the amount of knowledge that two or more applications have to share. Every component only has to connect to one endpoint. This also allows for a more efficient CPU usage by using asynchronous messaging. Application A can just send the message and forget. Giving it more time to work on new tasks. The persistency, reliability and routing will be handled by the messaging service which provides the core of the integration solution. [HW04]

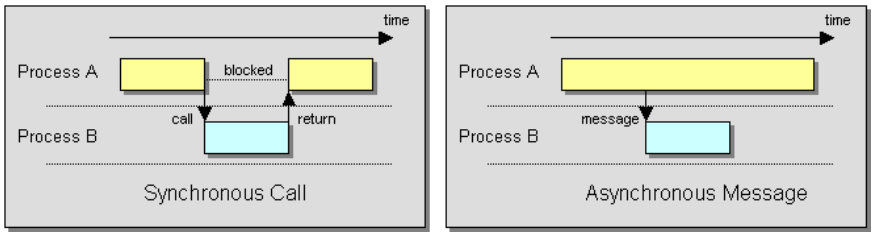

Figure 3: Synchronous and Asynchronous Call Semantics
Source: Hohpe, Woolf: Enterprise integration patterns[HW04]

Using such a messaging solution can also increase security. Instead of allowing communication between all applications, the network can be setup up in such a way only communication between each application and the messaging service is allowed. This shifts potential security risk to the messaging service. So the messaging service has to provide a central secure communication infrastructure. [QCP+14]
All these requirements lead to the enterprise application integration (EAI) paradigms with its pattern. Besides these enterprise integration patterns (EIP) it still can be hard to implement your own integration framework. Developers for an integration framework have to overcome the lack of control over the application to be integrated, provide a platform independent solution that can understand and translate between hundreds of different protocols. As a result the usage of an existing integration framework minimizes a lot of work. [IA18]
Further reading will go into how some existing integration frameworks solved these challenges using EIP.

2.3 Drawbacks of an Enterprise Service Bus

Like any separate application an ESB causes extra cost and time. An integration framework is an addition piece of software that has to be developed and maintained. A company that wants to use an ESB has to ether hire developers that already know how to use integration frameworks or pay their existing developers learn these.
Depending on the implementation of the integration framework it may introduces a single

point of failure to the entire service infrastructure of a company. In case all the routing, transmitting and transformation is done in a single application it introduces the risk that all other applications go down if the central communication hub fails. Consequently a decentralized integration solution is needed or a reliable backup solution for a central solution. By adding additional applications and abstracting the communication extra overhead gets involved. This impacts the size of the overall data and its transaction time. The messaging solution has to receive a message, read it, route it, maybe transform it and send it. This is a serious drawback especially for time sensitive applications. [HW04]

3 Integration Frameworks

3.1 How does it work?

Integration frameworks, either custom made or free generic ones, provide solutions to realize an ESB. The design of an integration framework is often based on enterprise integration patterns. The next section provides an example of an EIP starting with the initial problem and ending with the resulting solutions-patterns.

3.1.1 Assumptions on other applications

Software based integration solutions reduce the amount of assumptions one system makes over the other and therefore provides loose coupling instead of tight coupling. Even in a small system numerous assumptions are made.
Consider this small example: The web-front-end of a shop asks the backend if the article with the *productID: 4242* is in stock. The minimal protocol that both systems have in common is TCP/IP. This example already has several problems by making the following assumptions[HW04]:

- Both applications have to be on the same platform technology
- They have to share the same data-format
- The front-end has to know the address of the backend
- Both have to be available at the same time

Going through these assumptions shows that these are not guaranteed to be fulfilled. Network technology is on the move to IPv6. There might be the possibility that one application only can work with IPv4 or IPv6. [CKK13] Even besides that there is a little endian representation of 4242 resulting in `1000010010010` and `100100100001` for big endian encoding which is the number 2337. The same goes with the integer representation of any number. An application that reads a 64-bit integer will read an entirely different number than the

application that send 32-bit integer. As the other assumptions are quite clear, section 3.1.2 shows some solutions provided by integration frameworks.

3.1.2 Loose coupling

To overcome the problem of platform depending data-formats integration frameworks use self-describing and platform independent data structures like XML or JSON. These interchangeable formats are widespread and well adopted in many programming languages. [NPRI09] The actual information to share is encapsulated in a `message`. A message consists of a payload and optional parts like a header. The payload contains the actual information. The message is than send to a `channel`. A channel is an additional abstraction layer provided by the messaging system. Its actual implementation is up to the integration framework. The underlying transportation layer can be the IP protocol or any other transportation technology. [HW04]
Using channels avoids the need to know the location or address of the receiver. Additional routing and data transformation can then be done by the message system. Usually integration solutions provide multiple channels. Reusing the shop example, possible channel names can be *orders* or *invoice*. [HW04]
This allows the receiving application to pull messages from a channel whenever it wants. Resulting in the reduction of another assumption. The messaging system queue's message so that the sender and receive don't have to be available at the same time. This results in the possibility to pull messages at optimal speed. The receiver doesn't have to actively wait for a message or loses any messages because of an overload. [HW04]

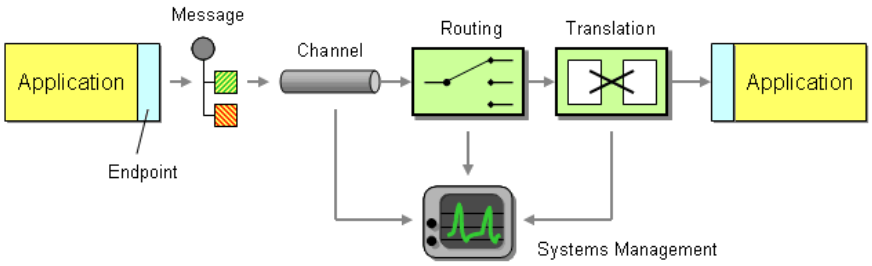

Figure 4: Basic Elements of an Integration Solution
Source: Hohpe, Woolf: Enterprise integration patterns[HW04]

Figure 4 shows a schematic overview of a simple integration solution. The blue box on the end of the first application shape is a `Chanel Adapter` and the second is an `Endpoint`. Most applications don't support the message-system of the integration framework by default. Implementing these functions directly in the application might not be the best solution. This would add additional complexity to the application and require developers to understand the application and the messaging-system. Applications from third-party vendors often do

not let you change the source code or view the source at all. Writing an adapter that converts the messages to the native application API solves this problem. Most integration frameworks provide templates for channel adapters and endpoints; therefore developers only have to know the API of the destination application instead of its full functionality. [HW04]

3.2 Advantages of integration Frameworks

Integration frameworks can help to split up the overall enterprise logic into smaller, modularized and independent parts. This gives a greater overview, more control over the individual components and reduces unwanted side effects for changes. It also provides all the advantages of loose coupling. It makes applications less depending on a specific platform, language or operating system. This provides the ability to replace business components with alternative services. [HW04]
The next section will show some example components of an integration framework and present some challenges of building and using an integration framework.

3.3 Requirements of integration Frameworks

To use an integration framework there are some conditions. The applications that should be integrated must provide a way to do so. If the overall business logic only consists of one component or of tightly coupled components than an integration framework will not solve all problems. Every application has to provide a way to individual call and use them. [Cum02]
If these conditions are fulfilled, companies have to decide to develop their own proprietary solution or to use existing ones.
Using an existing integration framework may come with license cost or less freedom in platform choice. But it also can save considerable amount of work.
The *Apache Camel* framework for example comes with a wide range of already implemented connectors for external services. Table 1 shows only a small part of over 280 Camel components. Specifying, implementing and testing all these adapters itself is a lot of work. Additionally, vendors may change their API, so the integration framework developers have to update their adapters. All this isn't even the main part of an integration framework. The messaging-system, routing, transformation and filtering between these adapters is the main purpose of an integration framework. [IA18]
The next section will present some common integration framework for enterprise and consumer usage.

Components
Azure
DigitalOcean
Docker
Dropbox
Facebook
FTP
Git
Google
GraphQL
Kubernetes
REST
SSH
Twitter

Table 1: Camel components

4 Overview of solutions

The following section will give a short overview of different integration frameworks aimed for enterprise and consumer usage.

4.1 In enterprise area

Spring Integration

The Spring Integration Framework is an extension of the `Spring-Framework`. It supports developer to use the Plain Old Java Object (POJO) concept. The POJO helps to reduce dependencies within a module to allow loose coupling. It provides the abstraction of channels, filters, and transformers. Some common endpoint interfaces like Representational State Transfer (REST) are already included. [spr19]

Apache Camel

Apache Camel is an open source integration framework by the Apache Software Foundation. [cam19] It is written in Java and implements most of the enterprise integration patterns. It is based on the patterns described in *Enterprise Integrating Patterns* by Gregor Hohpe and Bobby Woolf. Apache Camel is a well-known and widely adopted framework. Thanks to its open source nature, camel already provides many components and functions. Every company or individual developer can contribute to Camel and provide his knowledge to others. This makes Camel cover a wide range of requirements and rapidly adopt new technologies. It also forms the base for other frameworks. [IA18]
In section 5 are some examples of integration using Apache Camel.

Fuse ESB

Fuse ESB is an integration framework by Red Hat. It is based on Apache Camel and extends its functionality by proprietary Red Hat services. It focuses on distributed integration and cloud infrastructure. Fuse ESB provides on-premise deployment and official support by Red Hat. [fus19]

Mule

Mule is another ESB and integration framework developed by MuleSoft. It is a more application specific approach aiming for SAP, Salesforce and other enterprise specific solutions. Nevertheless, it also supports interaction with webservices. The whole product is more likely to be seen as an integration service instead of a pure framework. [mul18]

4.2 In consumer area

FLOW

FLOW is a service from Microsoft to connect webservices from different companies. The user can create own sequences of events. This is achieved by a connector for every service and a trigger. Every triggered sequence than can create several events. An example Flow can be the automatic creation of an appointment and a shared OneDrive folder if person XYZ sends an email. [flo19]

IFTTT

IFTTT is short for If This Then That. It is a free web service by IFTTT Inc. It has almost all the functionality of FLOW but extends it by providing additional features for IoT and Smart-Home devices. It works with Amazon Alexa, Google Assistant, Cortana and several of other Smart-Home devices from various companies. If This Then That also provides an enterprise service for businesses to directly integrate IFTTT into their products. [ift19]

Zapier

Zapier is a direct competitor of IFTTT. In contrast to IFTTT Zapier is more business orientated. It provides a much larger set of supported service (1500 different apps) but lacks support for Smart-Home devices. Zapier offers a greater customizability where, by contrast IFTTT aims for simplicity. [zap19]

5 Examples of Integration

This section will go deeper into the usage of integration frameworks by using the Apache Camel framework as an example.

5.1 Concepts of Camel

Apache Camel implements most of the enterprise integration patterns. It uses a message orientated middleware and a Domain-specific language (DSL) for routing rules. For every application a `CamelContext` is need. It is the base of every Camel instance. Figure 5 shows a `CamelContext` including routes and transformers. The `CamelContext` connects to the outer world with different components. Table 1 shows a small selection of these components. Camel can be used as a centralized or distributed integration framework. One application can handle all routing rules or every rule can have its own `CamelContext`. [IA18] Starting development with Camel is facilitated by Maven dependencies. Maven automatically resolves all dependencies, downloads the needed libraries and sets up the `classpath`. Alternatively, Apache offers precompiled JAR's. Camel can be integrated within every

java software or act as an independent application. On September 28th Apache Camel 3.0 was released. This release adds support for newer Java versions (Java 11) and modularized core parts of the library. [cam19]
The next section shows examples of DSL and some components.

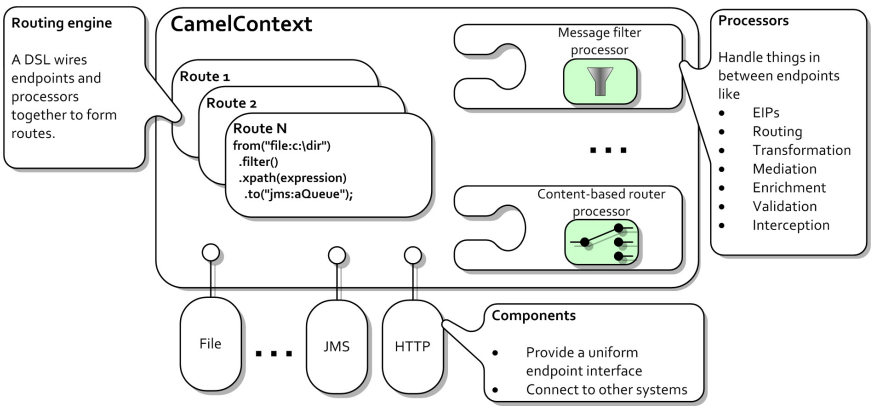

Figure 5: Architecture of Apache Camel
Source: Ibsen, Anstey: Camel in Action[IA18]

5.2 Examples in Camel

5.2.1 FTP-File Transfer

Listing 1 shows a minimal example of a Camel application. First the base `CamelContext` is needed. Then a route is added to the context. The endpoint instances are automatically generated. This example connects to a SFTP Server and copy's every new file to the folder „out". Every route allows several destinations. Developers just have to add another `.to` and an endpoint. With `context.start()` the `CamelContext` starts in a new Thread.

```
1  public static void main( String[] args ) throws Exception {
2      CamelContext context = new DefaultCamelContext();
3      context.addRoutes(new RouteBuilder() {
4          public void configure() throws Exception {
5              from("sftp:IP:PORT/PATH?username=USER&password=**").to("file:out");
6          }
7      });
8      context.start();
9      Thread.sleep(10000);
10     context.stop();
11 }
```

Listing 1: Camel FTP-File Transfer

5.2.2 Twitter searching and filtering

Listing 2 shows an example of a Camel Twitter application. The Camel application connects to Twitter via a built-in component and converts the tweets to Camels internal message format. Destination for the messages is a local WebSocket sunning on port 9090. Within the routing every message is converted by a custom converter that replaces the American date format with the German format. The converter is shown in listing 3. The converter could also be used to filter offensive language or to log the tweets to a file.

```
public static void main(String[] args) throws Exception {

    CamelContext context = new DefaultCamelContext();
    context.addRoutes(new RouteBuilder() {
        public void configure() throws Exception {
            //WebSocket setup
            WebsocketComponent wc = getContext().getComponent("websocket",
                WebsocketComponent.class);
            wc.setPort(9090);
            wc.setStaticResources("classpath:.");
            //Twitter setup
            TwitterSearchComponent tc = getContext().getComponent("twitter-
                search", TwitterSearchComponent.class);
            tc.setAccessToken(accessToken);
            tc.setAccessTokenSecret(accessTokenSecret);
            tc.setConsumerKey(consumerKey);
            tc.setConsumerSecret(consumerSecret);
            //Route
            fromF("twitter-search://%s?delay=%s", search, delay)
                //Convert data
                .process(new MyConverter())
                // and push tweets to web socket
                .to("websocket:camel-tweet?sendToAll=true");
        }
    });
    context.start();
}
```

Listing 2: Camel Twitter Connection

```
public class MyConverter implements Processor {
    public void process(Exchange exchange) throws Exception {

        String data = exchange.getIn().getBody(String.class);
        //Twitter Date format: Sun Dec 08 23:51:03 CET 2019
        String sub = data.substring(4,19) + data.substring(23,28);
        String msg = data.substring(29);
```

```
8           Date date = new SimpleDateFormat("MMM dd HH:mm:ss yyyy", Locale.GERMAN
                ).parse(sub);
9
10          //Use German date format and replace message body
11          String new_msg = new SimpleDateFormat().format(date) + " " + msg;
12          exchange.getIn().setBody(new_msg);
13      }
14  }
```

Listing 3: Camel Converter

5.2.3 Many more possibilities

There are thousands of more possibilities to use Camel to connect different software products. Camel allows to parse and cumulate logs, connect to different machines via `ssh` or generate WebHooks for every website.
The whole Apache Camel project can be found at `https://github.com/apache/camel` with several more examples. The book Camel in Action by Claus Ibsen and Jonathan Anstey is one of the official books to get started with Camel.

6 Conclusion

The examples in section 5.2 show how easy it is to start with integration framework. These frameworks eliminate the need to study, use and test the official API of every application that my worth integrating. It also reduces the amount of code to write and help to obtain better code quality. The impeccable functionality of an API call by an integration framework is much more likely than thousands of independent API calls of various developers. Integration frameworks also save money by allowing to reuse applications. There is no need to rewrite an application component because it doesn't work on a specific platform. It just can be integrated.
Integration Frameworks may also increase market competition. Enterprises and consumers can choose the best application for their needs and integrate it. They are not restricted by any specific platforms. This allows developers and companies to cherry-pick from many software components and integrate them into their business logic.
The Enterprise sector rapidly adapts to microservices, distributed computing along cloud and web technologies. There is a need for integration to manage the growing complexity in software. The same goes for growth of consumer devices for IoT and Smart-Home. To create a collaborative Smart-Home solution there is the need for internal standard or solutions that provide integration.

7 List of Tables

8 List of Figures

9 Listings

References

[cam19] Apache camel. `https://camel.apache.org/`, 2019. Accessed: 2019.12.02.

[CKK13] Deka Ganesh Chandra, Margaret Kathing, and Das Prashanta Kumar. A comparative study on ipv4 and ipv6. In *2013 International Conference on Communication Systems and Network Technologies*, pages 286–289. IEEE, 2013.

[Com18] Wikimedia Commons. File:esb.svg — wikimedia commons, the free media repository, 2018. [Online; accessed 14-December-2019].

[Cum02] Fred A Cummins. *Enterprise integration: an architecture for enterprise application and systems integration*. John Wiley & Sons, Inc., 2002.

[flo19] Flow webpage. `https://flow.microsoft.com/en-us/`, 2019. Accessed: 2019.12.02.

[fus19] Fuse esb. `https://www.redhat.com/en/technologies/jboss-middleware/fuse`, 2019. Accessed: 2019.12.02.

[HW04] Gregor Hohpe and Bobby Woolf. *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley Professional, 2004.

[IA18] Claus Ibsen and Jonathan Anstey. *Camel in action*. Manning Publications Co., 2018.

[ift19] Ifttt webpage. `https://ifttt.com/discover`, 2019. Accessed: 2019.12.02.

[IS18] Kasun Indrasiri and Prabath Siriwardena. Microservices for the enterprise. *Apress, Berkeley*, 2018.

[K+03] Patrick Kinney et al. Zigbee technology: Wireless control that simply works. In *Communications design conference*, volume 2, pages 1–7, 2003.

[MBF99] Michael Mattsson, Jan Bosch, and Mohamed E Fayad. Framework integration problems, causes, solutions. *Communications of the ACM*, 42(10):80–87, 1999.

[Men07] Falko Menge. Enterprise service bus. In *Free and open source software conference*, volume 2, pages 1–6, 2007.

[mul18] Mule. `https://www.mulesoft.com/`, 2018. Accessed: 2019.12.02.

[NPRI09] Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, and Clemente Izurieta. Comparison of json and xml data interchange formats: a case study. *Caine*, 9:157–162, 2009.

[QCP+14] Dino Quintero, Luis Carlos Cruz, Ricardo Machado Picone, Dusan Smolej, Daniel de Souza Casali, Gheorghe Tudor, Joanna Wong, et al. *IBM Platform Computing Solutions Reference Architectures and Best Practices*. IBM Redbooks, 2014.

[spr19] Spring integration. `https://spring.io/projects/spring-integration`, 2019. Accessed: 2019.12.02.

[Tan09] Andrew S Tanenbaum. *Moderne Betriebssysteme*. Pearson Deutschland GmbH, 2009.

[YLL18] Heetae Yang, Wonji Lee, and Hwansoo Lee. Iot smart home adoption: the importance of proper level automation. *Journal of Sensors*, 2018, 2018.

[Z302] Konrad Zuse archive. `http://www.konrad-zuse.net/main.html`, 2002. Accessed: 2019.11.24.

[zap19] Zapier webpage. `https://zapier.com/apps/integrations`, 2019. Accessed: 2019.12.02.

Declaration of Authorship

I hereby declare that the paper submitted is my own unaided work. I assure that I wrote this paper without using any other means and sources than those specified. As well as the sources used literally or analogously taken from the sources identified as such.

Signature (Henrik Gerdes)

Osnabrück, the February 25, 2020

A Appendix