Fraunhofer
ISST

FRAUNHOFER-INSTITUT FÜR SOFTWARE- UND SYSTEMTECHNIK, ISST

SILICON ECONOMY

B2B PLATFORM ECONOMY REFERENCE ARCHITECTURE CONCEPT

Report

Steffen Biehs

Martin Fitzke

Fraunhofer Institute for Software and Systems Engineering ISST

Carina Culotta

Timo Erler

Fraunhofer Institute for Material Flow and Logistics IML

Dortmund
January 2025

Table of Content

1 Introduction

Digital platforms are an emerging business model for B2B companies especially in logistics. Digital platforms can be described from a market-oriented perspective as enablers for market transactions between different parties. The digital platform reduces transaction costs such as search- or negotiation costs. The platform participants represent different market sides such as suppliers and buyers. Thereby, digital platforms benefit from direct and indirect network effects. Direct network effects emerge when users benefit from the presence of the same user group – a useful example is social networks where users benefit from other users with whom they can connect. Indirect network effects emerge when the market sides benefit from each other or so-called complementary providers. Complementary providers can offer additional solutions that enhance the value of the original products and services.

From a technological perspective, digital platforms can be seen as a modular, expendable technological infrastructure representing and depicting the respective business processes. Digital platforms manage and orchestrate the information flows about physical and virtual assets and financial streams and create benefits by interpreting and valorizing the generated data.

Many industries in the B2B domain have tried to implement digital platforms, seeing the potential for reducing transaction costs and creating new, digital solutions that increase efficiency. However, unlike their counterparts in the B2C or C2C domain, B2B-driven platforms fail to scale to the same level.

This is accounted for by the high complexity of industrial processes, the resulting lack of standards, and different levels of digitalization among the supply chain partners. Taking these challenges into account the Silicon Economy Initiative[1] was originated. Funded by the Federal Ministry for Digital and Transport between 2019 and 2024 more than 150 researchers from three different research institutes (Fraunhofer-Institute for Material Flow and Logistics, Fraunhofer-Institute for Software and Systems Engineering, and TU Dortmund University) have created more than 50 open-source software- and hardware solutions[2], as well as qualitative concepts that help companies implement digital B2B platforms.

Starting from the domain of logistics, the idea is to create a federal, B2B-driven platform economy that allows companies to realize their own scalable digital B2B platform solutions. In the "Silicon Economy Integration Guideline"[3] interested readers find an extensive overview of the Silicon Economy and how to create B2B-driven business models. Amongst others, a Platform Alignment Canvas has been created to help companies identify and describe their respective business models giving insights on how to use the various open-source software components provided by the Silicon Economy from a business perspective. Beyond the different open-source software components that offer different applications ranging from the digitalization of freight documents to AI-based yard management systems, a catalog of standard functions for

[1] www.silicon-economy.de

[2] The open-source software solutions can be found here: https://git.openlogisticsfoundation.org/explore/groups

[3] www.silicon-economy.com/integration-guideline

logistics[1] has been created, too. The standard functions help companies to understand and depict logistical processes (e.g., pallet exchange) and define the necessary data input. The data input can come from the Silicon Economy open-source and hardware software components. However, how to orchestrate and manage these data inputs along the different supply chain processes and partners in a scalable way is demonstrated by the underlying reference architecture.

In the following, the necessary basics concerning data marketplaces and the data market framework of the IDS are laid out. Following that an arc42[2] software architecture documentation is presented, and the requirements, the stakeholders, the necessary components for a data marketplace, and a suitable architecture for realizing B2B platforms within the Silicon Economy are explained.

[1] www.silicon-economy.com/standardfunktionen

[2] https://arc42.org/

2
Reference Software Architecture

This chapter presents the reference software architecture for a digital platform in the B2B context within the domain of logistics along a so-called fourth-party logistics service provider use case. The description is based on the arc42 documentation for software architectures.

2.1 Introduction and Goals

To ensure the practical relevance of the underlying architecture for a B2B-driven digital platform solution a use case was created that helps to design the respective architecture and ease understanding which data streams must be shared respectively rendered possible. In specific, a so-called fourth-party logistics (4PL) service provider was considered. A 4PL manages and orchestrates logistical processes without having own assets such as trucks or warehouses. Via a digital platform solution all relevant stakeholders such as an original equipment manufacturer, third-party logistics (3PL) service providers and second-party logistics (2PL) providers as well as suppliers and end-customers can be integrated.

The 4PL thereby provides a platform solution. However, all clients require easy and scalable integration. For the market participants, all other partners must be integrated in the same way so that data and information can be shared within the supply chain (e.g., status information or estimated time of arrival). The goal of the underlying reference architecture is to show how a digital platform solution can be designed in such a way that all actors can be integrated easily in a scalable way. Moreover, all necessary actors need to be able to communicate with each other and provide their digital services and information.

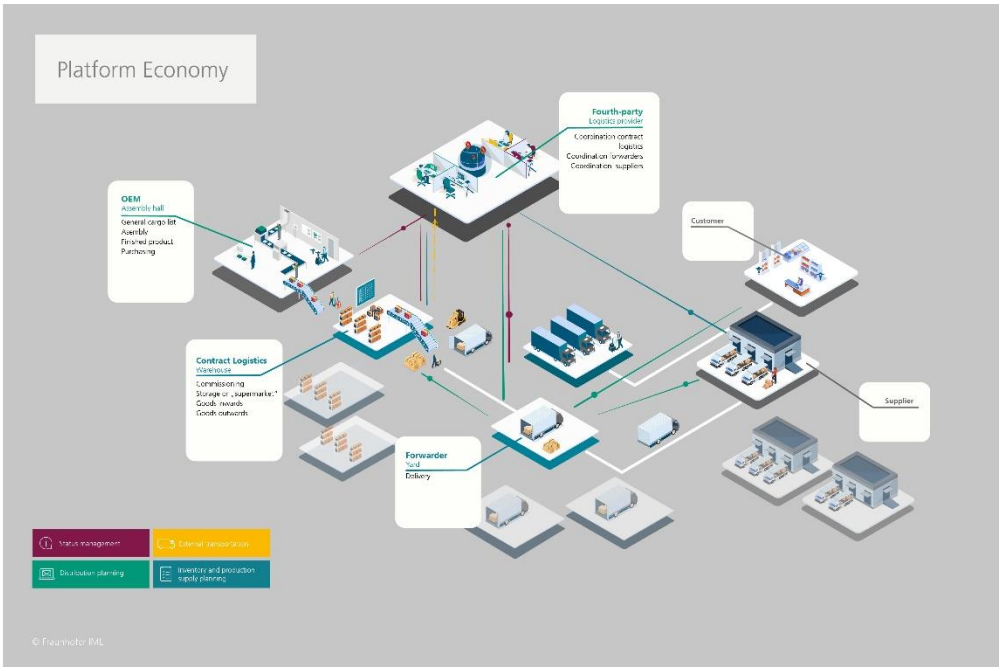

Figure 1: Use Case of a 4PL platform business model with relevant stakeholders

2.1.1 Requirements Overview

The requirements for the reference architecture were retrieved through several workshops of the project team. Following a domain-driven design approach, all stakeholders in the use case depicted above were highlighted and assigned with the data they must share. In this way a general understanding of how and to what extent data has the be shared and managed was retrieved laying the basis for the reference architecture. These results were also challenged by two different industry partners that currently pursue a 4PL business model.

The respective workshops highlighted the need for easy integration of various partners along different existing information systems (such as warehouse management systems or transportation management systems). In addition, data exchange must be secured and must allow for the sovereignty of the respective partners. Likewise, the different partners must be found and related to the relevant other partners (e.g., the 3PL with the 4PL). Finally, to ensure the scalability of the platform and relate to the business reality of the stakeholders it must be possible to easily exchange partners or remove partners from the business ecosystem if e.g., a supplier is exchanged.

2.1.2 Quality Goals

Based on the ISO 25010 standard, the following three quality goals were selected for the reference architecture:

Table 1 Quality goals

Quality Goal	Description
Operability	The reference architecture should be easy to learn so that it can be integrated into your own environment. It should help you get started in the platform economy
Compatibility	The reference architecture is to be used in existing IT landscapes. Integration must be possible with different architectures and technologies. In addition, IT landscapes of different companies should be able to be linked with this approach.
Transferability	It should be possible to adapt the reference architecture to the respective company scenarios and port it to the corresponding requirements of the company network.

2.1.3 Stakeholders

Within the reference architecture there are seven key stakeholder groups, which are described below. The stakeholder roles emerge from the underlying use case.

2.1.3.1 Participant

There is a generic participant who can assume different roles. This participant is a place holder for all stakeholders that can successively be integrated into the platform ecosystem such as new logistic service providers or new suppliers and customers.

2.1.3.2 4PL service provider

The 4PL service provider is the owner of the platform. He is responsible for the platform infrastructure and for the depiction of the relevant business processes. In specific, logistical services must be managed for the OEM – his prime customer. The OEM wishes to outsource logistical activities including activities on the premise of the OEM (warehousing, goods inwards, and goods outwards as well as packaging) as well as transportation activities (delivery to customers, pick-ups from suppliers). In specific the 4PL logistic service provider needs to acquire the following process information:

- Goods requested from the supplier
- Goods ordered from the supplier
- Goods dispatched by the supplier via the 2PL
- Goods received on the factory premises by the 3PL, picked and then handed over for assembly
- Goods completed by the OEM and sent to outgoing goods and then packed and dispatched again via the 3PL
- Goods sent via the 2PL to the customer
- Invoice paid by the customer

The digital platform solution offered by the 4PL is the "home" of all these information. At the same time, additional services such as purchasing activities and all other digital services such as status overview for the customer or estimated-time-or-arrival solutions, forecasting, and planning options can be offered by the digital platform accounting for the information transferred by all relevant partners. The 4PL service provider is the orchestrator of the entire process.

2.1.3.3 OEM

In the underlying use case, the original equipment manufacturer (OEM) is a manufacturing company. The OEM wishes to outsource its logistical activities as they are not part of the core business. At the same time, the OEM wants to expand manufacturing to other premises and therefore needs a service provider that can manage and orchestrate logistical activities beyond and across the different facilities.

2.1.3.4 Contract logistics (3PL)

The 3PL service provider is a contract logistics firm and the organizer of the logistical process. Whereas the 4PL service provider mainly is concerned with orchestration, the 3PL service provider is responsible for the operations. In the underlying use case, the 3PL service provider e.g., manages all activities on the premise such as the warehouse or the goods inward and outwards as well as packaging and providing the correct set of parts for assembly in the hall of the OEM. Likewise, the 3PL may manage also the second-party logistics (2PL) service providers directly.[1]

2.1.3.5 Forwarder (2PL)

[1] It is also possible that the 4PL directly manages the 2PL activities. This is dependent on the respective business logic.

The 2PL service provider is a forwarder that carries out the physical activities outside the premise of the OEM which includes transportation of the incoming and outcoming goods from suppliers respectively to customers. The 2PL is either managed by the 3PL or by the 4PL directly.

2.1.3.6 Supplier

The supplier supplies the OEM with the necessary parts. In fact, there are several suppliers across various locations.

2.1.3.7 Customer

The customer is the end-customer of the OEM. He receives the finished and final product.

2.2 Constraints

In the following, any requirements that constraint software architects in their freedom of design and implementation decisions or decision about the development process are documented. These constraints sometimes go beyond individual systems and are valid for whole organizations and companies.

2.2.1 Technical Constraint

Table 2 Technical constraints

Constraints	Description
Use of open-source dependencies	Only dependencies that are open-source and available with suitable licenses should be used.
Connector	The architecture will use an EDC Connector[1] and its extension feature as a technological base.
Sufficient code documentation	To increase future maintainability, the application source code should be documented with java-doc style documentation in addition to in-line comments in.

2.2.2 Conventions

Table 3 Conventions

Constraint	Description
Architecture Format	Based on the English version of the arc42 template.
Adaptability	The architecture should be developed in such a way that it can be easily adapted to any use cases that arise.

[1] https://eclipse-edc.github.io/documentation/ (16.12.2024)

2.3 Context and Scope

The underlying architecture is meant to fulfill the overall requirements of the Silicon Economy allowing for easy integration of partners, and the depiction of plural ecosystems. Moreover, this document is part of several working results of the Silicon Economy and complements the Silicon Economy Integration Guideline (see above) as well as the catalog of Logistical Standard Functions (see above). Thus, together with various solutions including the open-source soft- and hardware components (see above) as well as the qualitative results the reference architecture is meant to help companies implement their digital platform solutions.

2.4 Solution Strategy

The solution strategy is based on the approach of creating a business scenario coordinated with stakeholders. This business scenario (see above) is used on the one hand to create a proof-of-concept prototype and on the other hand to derive and validate the reference architecture. The resulting reference architecture is evaluated with the partners and continuously developed further.

2.5 Building Block View

This section describes the decomposition of the reference architecture into subsystems and modules. The building block view level 1 presents the first decomposition level of the reference architecture containing all subsystems including their interfaces. For the subsystem this overview also includes a more detailed breakdown into level 2 (see section 2.5.2).

2.5.1 Whitebox Overall System – Level 1

The reference architecture system breaks down into two kinds of subsystems as presented in Figure 2 and explained in Table 4. The dashed arrows represent connections between the subsystems and indicate that the two connected entities can send messages to each other. The squared boxes on the membrane of the system are interaction points with the outside world.

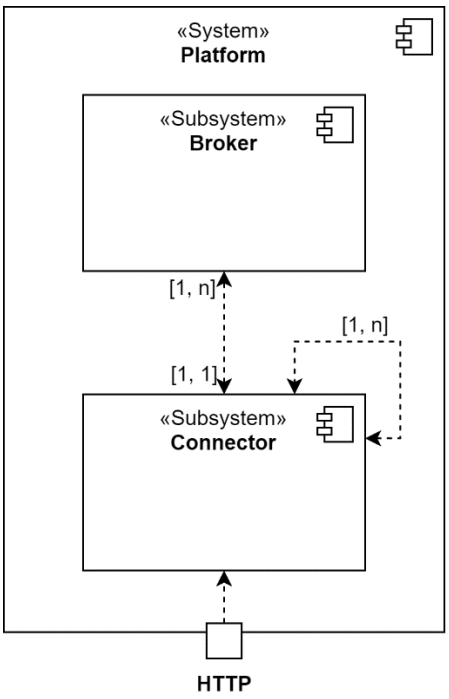

Figure 2: Reference Architecture, building block view, level 1.

Table 4: Description of platform subsystems

Subsystem	Short description
Connector	Base infrastructure that provides web server functionalities and an extension mechanism.
Broker	Connector where all platform services register for exchange of information and actions.

2.5.1.1 Blackbox Subsystem Connector

Purpose

An open-source project that is used as framework that provides an infrastructure for a webserver and an extension mechanism that allows to build services with a variety of different functionalities that are run in parallel and can also be isolated.

Interfaces

The Connector has an HTTP interface which allows interaction with the connector itself and the extensions that are integrated.

2.5.1.2 Blackbox Subsystem Broker

Purpose

A Connector that runs an extension where all services of a platform must register and provide metadata to. Other Connectors and services on the platform can then use this data to interact with each other.

Interfaces

The Broker has an HTTP interface to communicate with other Connectors/services.

2.5.2 Whitebox Level 2: Connector

As shown in Figure 3 and described in Table 5, the Connector component is divided into two modules. A bidirectional connection between subsystems is represented by the dashed line.

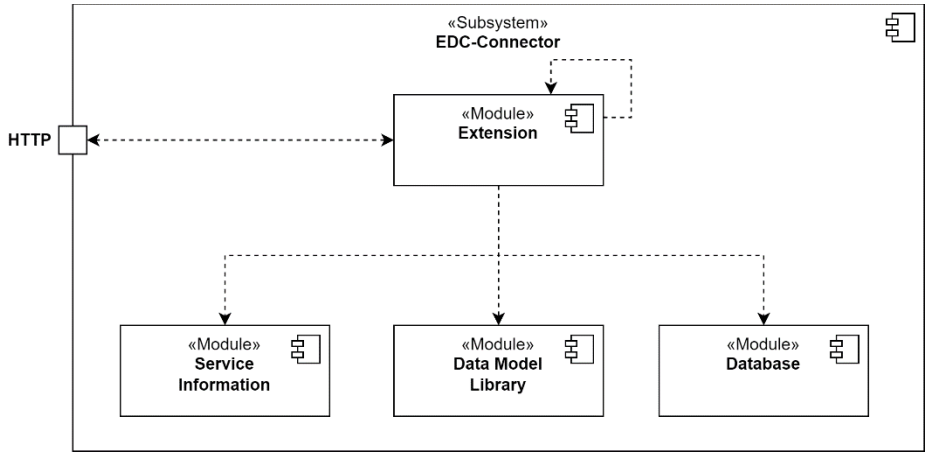

Figure 3: Whitebox view of Connector component.

Table 5: Description of Connector modules

Module	Short Description
Extension	A subsystem of a Connector that may offer an http-accessible REST-API, can contain arbitrary functionality, such as other extensions and libraries, and has access to the Connector's other systems at runtime.
Service Information	An extension that provides information about the running extensions and APIs of a Connector to external entities over a REST API and automatically registers the Connector it is run on at the platform Broker.
Database	An extension that contains an abstraction layer for an underlying database and can serve as a single point of access for database-related tasks for all extensions running inside the Connector. It also offers JSON parsing functionality at runtime.
Data Model Library	A library that contains the complete prototype data model as well as utility functions for JSON parsing and database interaction.

2.5.2.1 Blackbox Module Extension

Purpose

Provide an arbitrary new functionality to a Connector with possible utilization of other extensions and libraries while optionally offering a REST-API over HTTP.

Interfaces

An extension optionally offers a REST-API over HTTP and therefore communication to other services can be realized over HTTP.

2.5.2.2 Blackbox Module Service Information

Purpose

An extension to provide information about the Connector's extensions and APIs as well as registering at the platform Broker.

Interfaces

The extension offers a REST-API over HTTP.

2.5.2.3 Blackbox Module Database

Purpose

A single point of access for all extensions to interact with a certain database in the form of an extension that provides a database abstraction layer and JSON parsing functionality.

Interfaces

The Database extension provides a ready-to-use database with an API that exposes all required functions to interact with the database and parsing JSON data.

2.5.2.4 Blackbox Module Data Model Library

Purpose

Bundle the complete data model required for the prototype platform in a single library to simplify implementation and provide an abstraction for database interactions and JSON handling in general. This library provides the underlying functions used in the Database extension.

Interfaces

The exposed functions of this library are used for database interaction and JSON handling.

2.5.3 Whitebox Level 3: Extension

The Delivery Planning module is composed of two submodules, as indicated in Figure 4 and presented in Table 6.

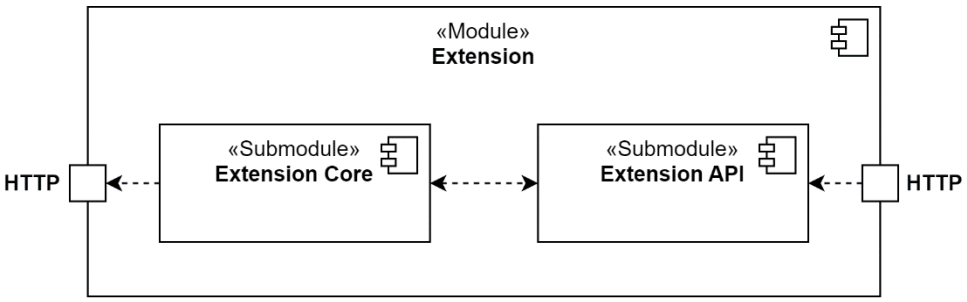

Figure 4: Whitebox view of an Extension.

Table 6: Description of Extension submodules

Submodule	Short Description
Extension Core	Contains the extension functionality that runs on the Connector and implements the API interface.
Extension API	The API interface of the extension that defines the REST-API.

2.5.3.1 Blackbox Submodule Extension Core

Purpose

The Extension Core contains the functionality that the extension provides at runtime.

Interfaces

The core implements the interface(s) defined in the Extension API module so that the REST-API endpoints have functionality. Additionally, the Extension Core has the possibility to connect to other services via HTTP.

2.5.3.2 Blackbox Submodule Extension API

Purpose

The Extension API module defines the interface from which the REST-API is generated. It can then be implemented by any core or extended by other APIs.

Interfaces

The Extension API defines the API, but the interface must be implemented to provide the endpoints at runtime.

2.5.4 Whitebox Level 4.1: Extension Core

The Extension submodule is composed into two classes, as indicated in Figure 4 and shown in Table 7.

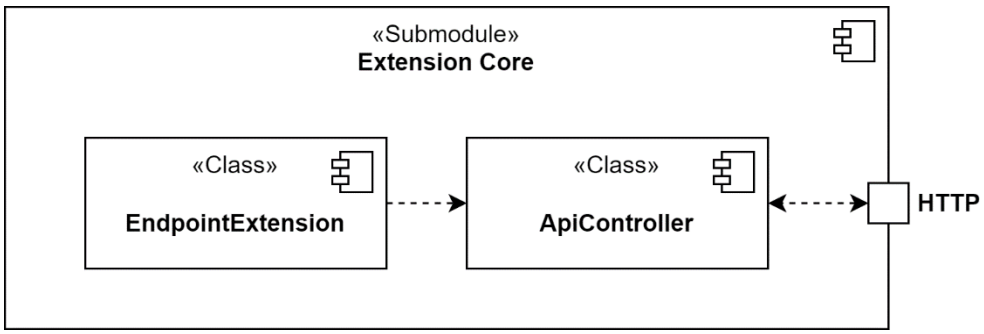

Figure 5: Whitebox view of the Extension Core submodule.

Table 7: Description of Extension classes

Class	Short Description
ApiController	Implements the API of the extension.
EndpointExtension	Initializes the Extension.

2.5.4.1 Blackbox Class ApiController

Purpose

Contains the implementation of the API.

Interfaces

Provides HTTP endpoints for external communication.

2.5.4.2 Blackbox Class EndpointExtension

Purpose

Contains the code that is called by the Connector to initialize the extension, including the API and web interface, as well as all other services that are required by the extension.

Interfaces

This class hat no interfaces.

2.5.5 Whitebox Level 4.2: Extension API

The Extension API submodule can be composed of multiple interface classes, as indicated in Figure 6 and described in Table 8.

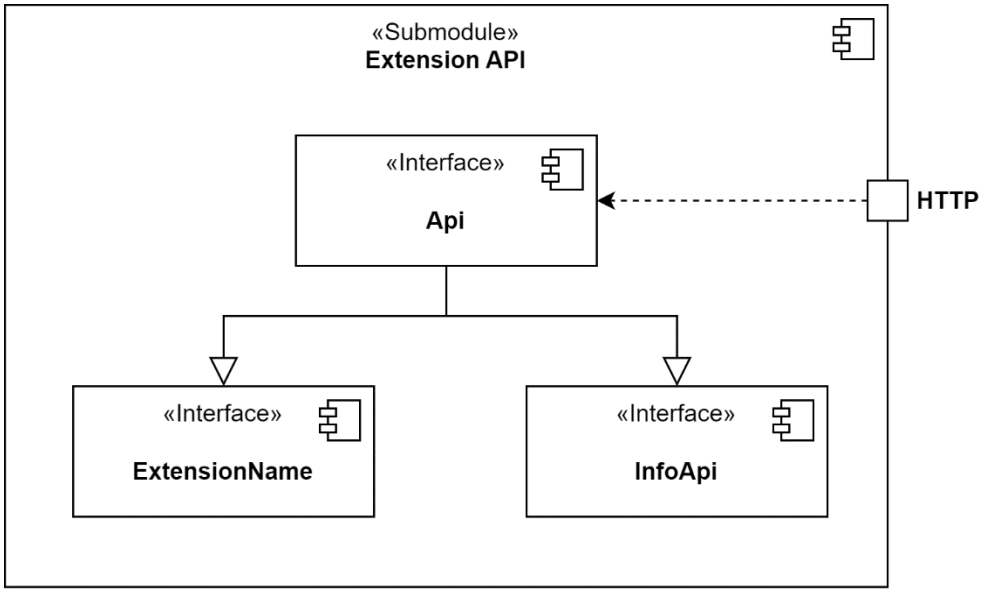

Figure 6: Whitebox view of the Extension API submodule.

Table 8: Description of Extension interfaces

Class	Short Description
Api	Main API of the extension.
InfoApi	API example with some common routes shared between extensions.
ExtensionName	Specialized API that represents the extension specific API calls.

2.5.5.1 Blackbox Class Api

Purpose

The main API definition from that also defines some metadata for the OpenApi generators. It may extend other interfaces and bundle them in this interface.

Interfaces

Contains all HTTP API endpoint definitions for an extension.

2.5.5.2 Blackbox Class InfoApi

Purpose

Provide some basic or common definitions that can be shared over many extensions.

Interfaces

Contains HTTP API endpoint definitions for an extension.

2.5.5.3 Blackbox Class ExtensionName

Purpose

Defines the actual extension interface with the functions/routes that are specific to this extension.

Interfaces

Contains HTTP API endpoints for an extension.

2.6 Runtime View

In contrast to the static building block view, this section visualizes dynamic aspects of the reference architecture. The runtime view describes concrete behavior and interactions of the system's building blocks across the architecture, i.e., how instances of building blocks within the reference architecture perform their job and communicate at runtime.

2.6.1 Register at Broker

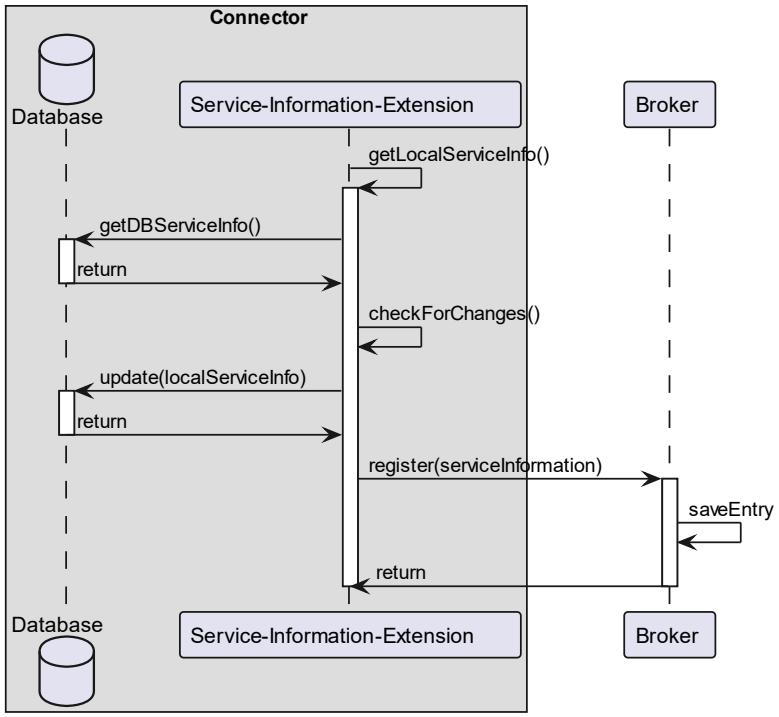

Figure 7: Runtime View for process Register at Broker

As depicted in Figure 7, to register at a service at the platform broker, the Service-Information-Extension of a Connector checks if information about the self-offered services is available locally (e.g. in a file) and then loads the stored information in the database. These two versions of the information are then compared and if the local version differs from the one stored in the database, the database version is updated. Following this, the extension then sends a registration request with meta information to the Broker. If the Broker already knows the service and the current information is correct, the Broker communicates that the service is already registered, if the service is new or something changed the Broker responds accordingly.

2.6.2 Request information from Broker

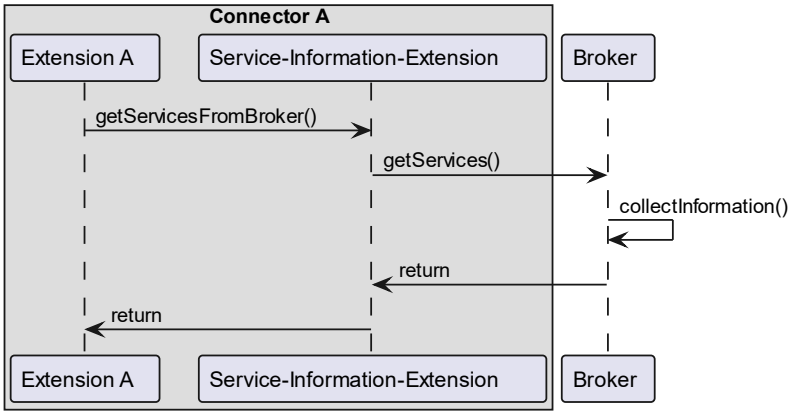

Figure 8: Runtime View for Request information from Broker

To request information about other services from the Broker the process is carried out as shown in Figure 8. Extension A initiates the sequence by requesting to obtain the information about the desired services from the Broker. The Service-Information-Extension receives this call and requests the information from the Broker.

The Broker then gathers the necessary data and returns the collected information to the Service-Information-Extension, which subsequently sends the result back to Extension A. This flow ensures that Extension A receives the most up-to-date service information.

2.6.3 Interaction between services

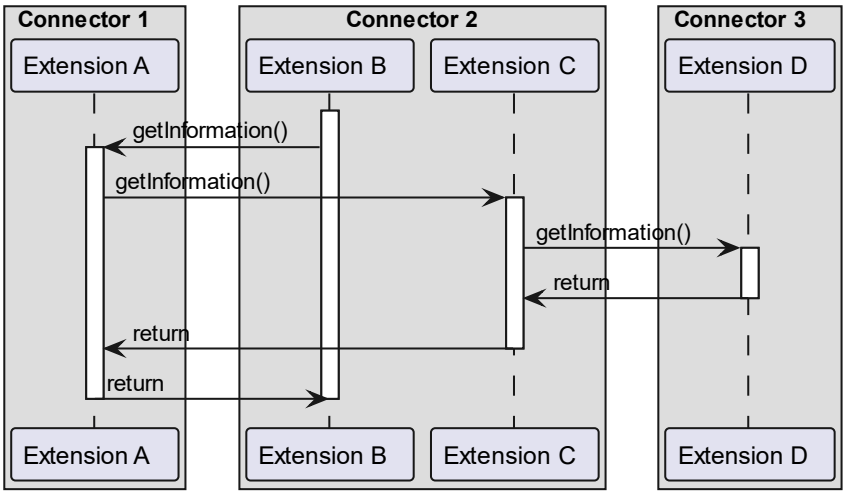

Figure 9: Runtime View for Interaction between services

The in Figure 9 depicted process shows the interaction between different services. The interactions can be linked and carried out arbitrarily between the different entities, so workflows can involve multiple Connectors and extensions if it is required. Here, Extension B of Connector 2 initiates the sequence by requesting information from Extension A of Connector 1. Upon receiving this call, Extension A processes the request and subsequently calls Extension C, as it requires information that Extension C provides. Extension C then makes a request to Extension D of Connector 3 with another piece of required information. As soon as Extension D returns the information all participants of this interaction can sequentially respond with the desired information and fulfill their received request and conclude the interaction.

2.7 Deployment View

Software does not run without hardware. This view describes the deployment of the reference architecture. It describes the technical infrastructure used to execute the reference architecture, with infrastructure elements like computing nodes and the mapping/installation of compiled software building block instances to that infrastructure elements.
The following deployment diagram Figure 10 shows the deployment of the system in a local environment.

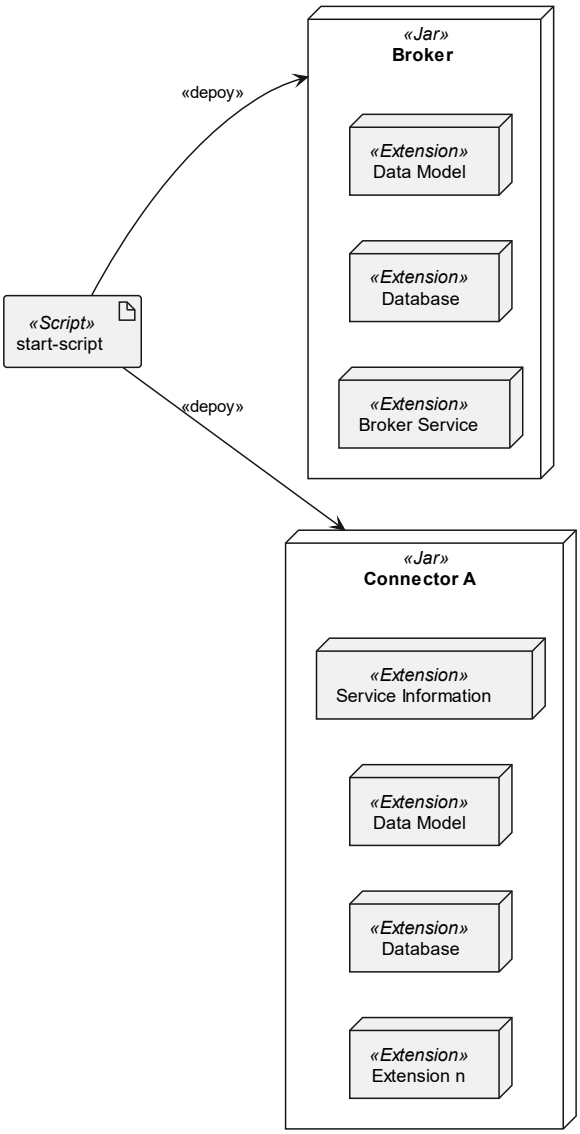

Figure 10: Deployment view

The script deploys two jar files. The first one is the Broker. The Broker contains an extension called *Broker Service*. This one gives the Broker the ability to store and provide meta data from different Connectors. The other extensions deliver basic functionalities for the Broker.
The second one is the Connector, which is here called Connector A. This represents every Connector that is deployed. A Connector also used the extensions Data Model and Database to get basic functionalities. Besides this, the Service Information extension is also used to send meta data information to the Broker. The last one, Extension n, stands for every extension that is deployed within this Connector.

2.8 Cross-Cutting Concepts

These concepts form the basis for conceptual integrity (consistency, homogeneity) of the architecture. Thus, they are an important contribution to achieving inner qualities. This section describes general structures and system-wide cross-cutting aspects. It also presents various technical solutions.

2.8.1 Data Model

A shared data model is required to allow a hassle-free communication between all services. The platform related models are re-usable and extendable and other models are use-case specific.

2.8.2 API

All extensions define their own REST-API over which the exposed functionalities are accessible with well-defined interface definitions.

2.8.3 Architectures

The platform is designed with a combination of the Peer-to-Peer, Client-Server, and Plugin architectures. A short description of their purpose is presented in the following sections.

2.8.3.1 Peer-to-Peer Architecture

The platform network setup is realized with a peer-to-peer architecture. Every Connector can communicate directly with any other Connector as soon it received the relevant information from the platform Broker.

2.8.3.2 Client-Server Architecture

The inter-Connector interaction is carried out by using the Client-Server architecture. The Connectors send requests over HTTP. The requests are sent and handled by extensions running inside the Connector. This allows the Connectors to be client and server at the same time in certain scenarios when different extensions interact with each other.

2.8.3.3 Plugin Architecture

The internal Connector architecture is realized using the Plugin Architecture and in this version the plugins are called extensions. It consists of a core holding the base functionality and a variety of extensions integrated out of the box. Additional extensions are used to realize the Connector services and they can also interact with each other.

2.8.4 Communication

The RESTful API communication is used between the different services and is based on HTTP. This allows for flexible, clear, and sufficiently quick communication and configuration of the services which are encapsuled in extensions.

2.9 Architecture Decisions

This section contains important, expensive, large scale or risky architecture decisions including rationales.

2.9.1 Using Template Connector

2.9.1.1 Problem

Every participant of the platform who wants to provide a service requires a personal Connector with a use-case tailored configuration and variety of extensions. The base setup is mostly equivalent.

2.9.1.2 Constraints

The implementation should be as simple and dynamic as possible.

2.9.1.3 Considered Alternatives

Building a complete setup from scratch for all participants.

2.9.1.4 Decision

A template Connector was created that offers a solid base configuration for all participants. The setup of extensions and configuration files are customized for the participants needs to fit their use-case.

2.9.2 Using Template Extension

2.9.2.1 Problem

For the interactions in the workflow different functionalities are required. These functionalities repeat in certain steps with different participants.

2.9.2.2 Constraints

The effort to realize repeating functionalities should be kept minimal.

2.9.2.3 Considered Alternatives

Creating one extension per Connector that encapsules all the services required functionalities.

2.9.2.4 Decision

These functionalities should be encapsuled inside extensions that can be used on every Connector. As a common base, a template extension was created. All developed extensions are based on this template.

2.10 Quality Requirements

The quality scenarios in this section depict the fundamental quality goals of the data marketplace as well as the quality goals defined earlier in this document. They allow the evaluation of decision alternatives. A listing of the quality requirements is shown in Table 9.

Table 9: List of quality requirements

ID	Scenario
1	Every extension has the same skeleton.
2	Every Connector has the same skeleton.

3	Each Connector has its own ready-to-use database.
4	Each extension uses the same data model.
5	The usage of this reference architecture is independent from technology
6	The communication is scalable
7	Systems in a network based on the reference architecture are exchangeable

2.11 Risks and Technical Debts

This section describes the risks which might occur when you use the architecture and how they are mitigated.

2.11.1 Risk 1: Versioning incompatibilities

2.11.1.1 Description

The EDC Connector features might change or get discontinued in new versions and that may result in an incompatibility with the architecture described in this document. The most significant feature required for the architecture is the extension mechanism.

2.11.1.2 Risk Mitigation

This can be prevented by using a technology stack that offers a framework for developing and combining extensions into a single service.

2.11.2 Risk 2: Setup up of Data Space components is not trivial

2.11.2.1 Description

Deploying Connectors and Data Space components is no trivial matter. Configuration and launching require a good technical understanding of the structure and functioning of the components.

2.11.2.2 Risk Mitigation

Participation in training courses or contributing to the development of components as part of open-source development vastly improves knowledge of how Data Space components work and can be deployed.

2.11.3 Risk 3: Connection to another Connector fails

2.11.3.1 Description

The connection to another Connector may be interrupted. In this case, no more data can be obtained between the Connectors.

2.11.3.2 Risk Mitigation

Mechanisms can inform when it is no longer possible to connect to another Connector.

2.12 Glossary

The most important domain and technical terms that your stakeholders use when discussing the system are described in Table 10.

Table 10: Glossary

Term	Explanation
Connector	The **Connector** is a basic component that offers a framework for HTTP based communication and an extension mechanic to build use case tailored processes
Broker	The **Broker** is an entity that has meta information about all involved parties and helps entities to reach each other and carry out interactions
HTTP	HTTP stands for **Hypertext Transfer Protocol**, which is a protocol used for transferring hypertext requests and information on the network.
API	API stands for **Application Programming Interface**, which is a set of rules that allows different software applications to communicate with each other.
REST-API	REST-API stands for **Representational State Transfer Application Programming Interface**, which is an architectural style that uses HTTP requests to access and manipulate data.

3 Conclusion

The underlying reference architecture highlights how companies can implement a digital platform solution using the logic of Brokers and IDS Connectors. That allows easy registration of services and simple detection of those services within the platform. The basic requirement for a successful B2B platform economy is the scalability of the system. Since supply chains in the industrial sector are characterized by high levels of heterogeneity in the participants' digital skills, simple integration solutions are required. Thus, companies that wish to build and implement their own platform solutions and wish for easy integration of their partners can use the underlying reference architecture and the respective logic as a starting point for their B2B platforms respectively as a starting point for digitalizing their supply chain processes.

Together with the different results of the Silicon Economy project, companies find valuable information and sources for digitalizing logistical and supply chain-related processes. Apart from various levels of digitalization skills, the lack of de facto standards is an additional challenge for companies in the B2B field. Especially in logistics, where processes are similar and aim for the same result (e.g., a transport or a pallet exchange or packaging), companies still have individual rules and ways to realize their processes. This hampers the harmonizing of digital solutions which again hampers the emergence of B2B platforms. Thus, de facto standards are necessary.

Those standards can be created by developing so-called commodity solutions jointly in an open-source process. This is the idea and purpose of the Open Logistics Foundation[1] that was brought to life during the project phase of the Silicon Economy and now operates as an industry-funded, independent organization. On the one hand, the Open Logistics Foundation hosts the Silicon Economy open-source software solutions, while on the other hand, different working groups advance the open-source developments. Within the different working groups industry representatives develop joint de facto standards and create different open-source software solutions for various logistical challenges. The Silicon Economy initiative has provided the starting point for this open-source movement within the B2B domain of logistics and beyond. Companies can benefit from the open-source software solutions and the idea of open-source being an innovation method as well as a cooperation- and collaboration tool. Openness, cooperation, and collaboration within the supply chain in an organizational matter but also on technological and software-related levels enables and eases B2B platform business models.

[1] https://openlogisticsfoundation.org/