CHARIOT - Description of the Decentralized Communication System Architecture

Deliverable 3.1

WP3: Communication



Version	0.1
Due Date	30.06.2017
Delivery Date	01.08.2017
Nature	Report

List of Figures

Table of Acronyms

Acronym	Meaning
API	Application Programming Interface
CoAP	Constrained Application Protocol
CS	Content Store
DNS	Domain Name System
DS	Directory Service
FIT	Forwarding Information Table
GUI	Graphical User Interface
ICN	Information Centric Networking
IoT	Internet of Things
JIAC	Java-based Intelligent Agent Componentware
JSON	JavaScript Object Notation
MQTT	Message Queue Telemetry Transport
M2M	Machine-to-Machine
NDN	Named Data Networking

OWL-S	Web Ontology Language for Semantic Web Services
PIT	Pending Interest Table
P2P	Peer-to-Peer
RE	Runtime Environment
REST	Representational State Transfer
SPARQL	SPARQL Protocol and RDF Query Language
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UI	User Interface
URI	Uniform Resource Identifier

Table of Contents

1. Introduction

This document aims to conceptualize the data communication infrastructure of the CHARIOT middleware, by explaining the main aspects of the directory service, and details of the communication between CHARIOT components. The communication layer in CHARIOT is responsible for establishing communication between devices and services in the service layer. The communication layer includes a directory service that stores device attributes and device location information together with descriptions of services that make use of device information. The directory service matches incoming service requests with relevant IoT entities and supports scalable communication between different runtime environments to enable P2P links between IoT devices. It is foreseeable that a centralized directory service cannot reach the desired performance and reliability for such a complex and dynamic IT infrastructure. For this reason, a decentralized and hierarchical directory service is to be developed in CHARIOT.

The data communication infrastructure consists of several communication interfaces between CHARIOT components, a distributed and scalable directory service and a semantic search component that enables searches inside the directory service. The architecture is based on state of the art approaches from similar projects and our technology partners of JIAC and IOLITE. The rest of the document is organized to present the features of CHARIOT communication infrastructure components. Communication interfaces and messaging formats are described in Section 2. Section 3 focuses on the distributed and scalable directory service architecture, and the semantic search query component is explained in detail in Section 4. The document is finalized with the conclusion section.

2. Communication Interfaces and Message Formats

The message format library contains application-independent formats for messages between devices, their respective device agents, and the directory service. In the architecture of CHARIOT, runtime environments can also be considered as IoT gateways that collect all information of devices and then forward this information to the service layer. For this reason, runtime environments handle messaging between devices and other CHARIOT components.

To establish an interconnection between entities that are connected to CHARIOT, messaging formats and communication interfaces need to be defined for runtime environment-to-agent communication, agent-to-directory service communication, agent-to-agent communication, and device-to-device communication. The message types between CHARIOT components in the communication layer are listed as follows:

1. Device Registration
2. Device Data Update
3. Device Data Request / Response
4. Device Removal
5. P2P communication
6. Other Message Types (i.e, messages that inform other components about errors and unknown messages)

Figure 1 displays the types of messages between the proxy messaging app and the communication layer, and Figure 2 visualizes these messages together with CHARIOT components.

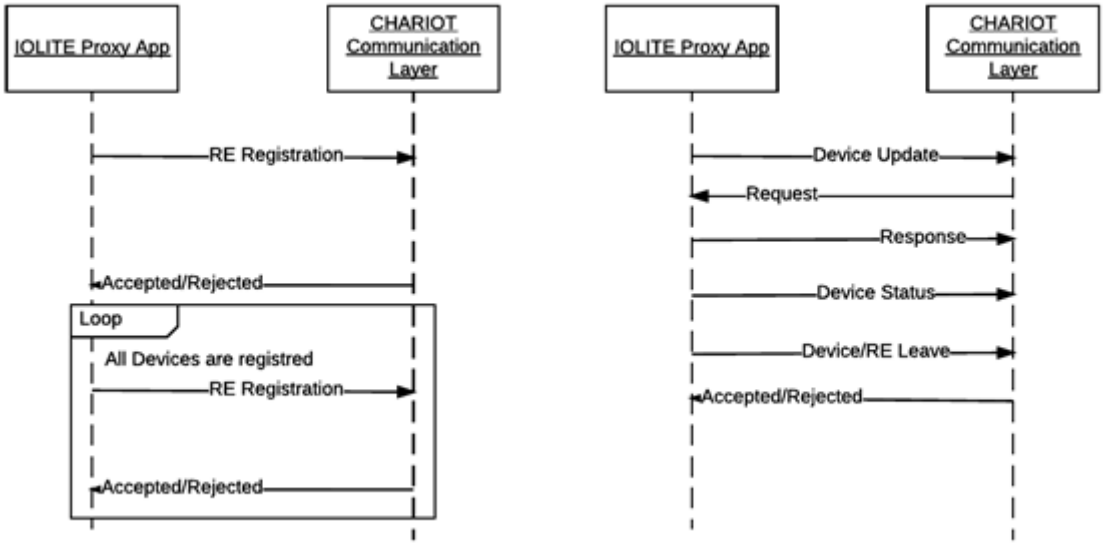

Figure 1: Runtime Environment and CHARIOT Communication Layer Message Exchange

In order to enable devices to communicate with each other and with the directory service, some components of the CHARIOT middleware need to take part in the communication process. These components are defined either in the device layer or in the service layer. In the device layer, runtime environments are responsible for establishing communication between devices. These runtime environments have their corresponding agents in the service layer to establish communication between devices and device agents, to register devices to directory service, and to update device data in the directory service and device agents. In order to establish all these messaging inside a runtime environment, we can list the required messaging interfaces in the CHARIOT middleware as follows:

1. Agent to Agent Communication: ActiveMQ
2. Agent to Directory Service Communication - Restful API
3. Runtime Environment to Agent Communication: Restful API
4. Device Registration and Data Update to Directory Service: Restful API

2.1 Device Registration

A registration request is used to connect devices to the directory service and device agents in the service layer. Before a device can be registered, the runtime environment has to register its own runtime environment agent to the service layer. Registrar agent is used to register a runtime environment to the service layer. A registration request is sent to the registrar agent and the runtime environment is authorized to set its own JIAC agent if it has valid credentials. After setting the runtime environment agent, the path for communication with the runtime environment is stored in the directory service. Registrar Agent runs as a server and listens to registration requests of runtime environments. The runtime environment and the registrar agent communicates by using RESTful API.

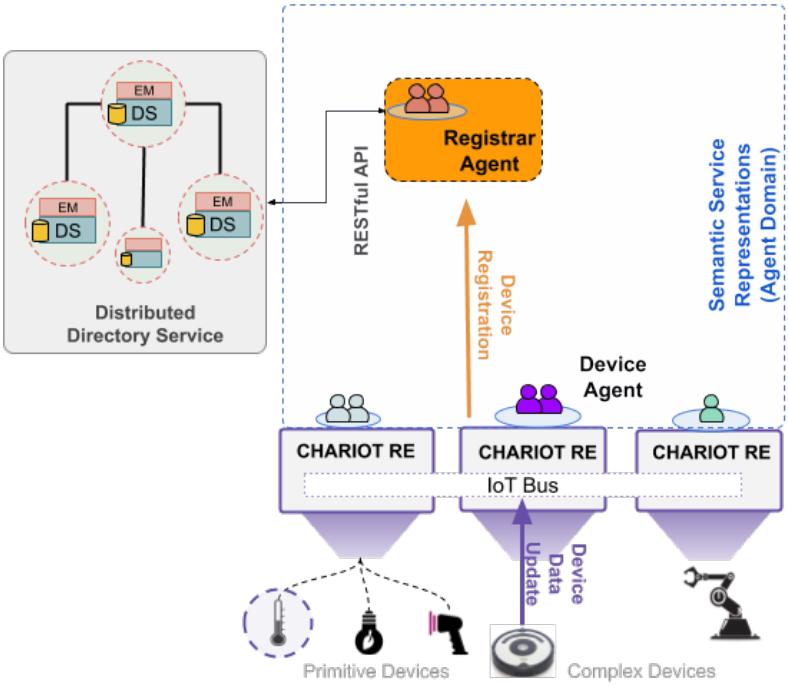

Figure 2: Communication Layer Modules and their Interactions

For device registration, runtime environment has a messaging application that collects all device driver information. Device information is first gathered at this messaging application inside the runtime environment and then sent to the runtime environment agent in the service layer. This application sends a registration request to runtime environment agent whenever a new device driver is added to the runtime environment. Device registration information is then stored as a service description URI in the directory service. After this registration process, the devices can send their data updates to their software entity in the service layer, called the device agent. In Figure 3, the sequence diagram for runtime environment registration and device registration are displayed.

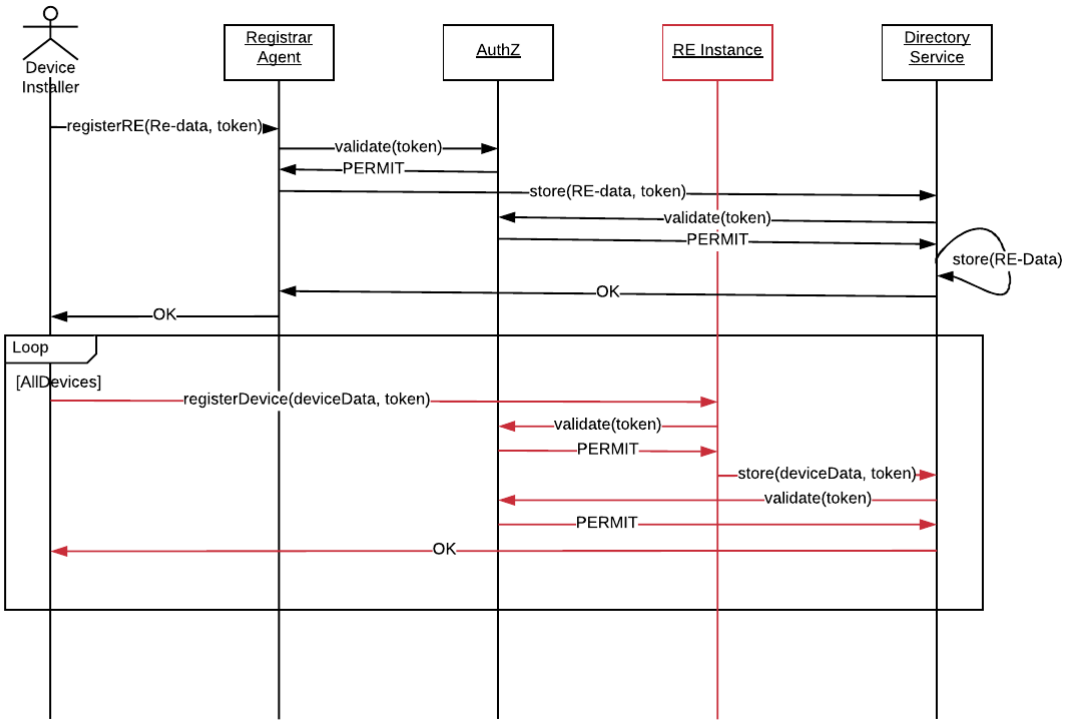

Figure 3: Sequence Diagram indicating the registration process and the communication with the responsible components along with the RE instance.

The content of the registration messages and the steps of the registration process are given below:

- **Registration Request:** Runtime environment messaging application collects all device information for registration. This application then sends a registration request to the registration agent. The IP, port and Agent ID information of the registrar agent is hard-coded to this application. Registrar agent runs as a server that listens to registration requests. When a request arrives, it then checks the security key to grant access to directory service. The message content of the registration request is given in Figure 4.

```
{
   message-type:”RE_Registration_Request”
   re-identifier:”RE-1”,
   location-name:”Warehouse”,
   SecurityKey: “SecurityKey”
   devices:[TempSensor1, TempSensor2, CarbonDioxide1, etc.]
   IP:”XXX.XXX.XXX.XXX” (for REST communication between agents and RE)
}
```

Figure 4: Registration Request Message Content

- **Device Data Information:** The RE messaging application sends device registration data to a registration agent in JSON format. This device data will then be used to create a service description in the directory service. The content of the message includes a

device property profile created by the runtime environment. The message content is given in Figure 5.

- **Binding the Device to its Device Agent:** The registration agent finds the respective device agent of the device and sends this property profile to this agent. Device data information in the property profile describes the device agent, and after a successful registration, the device is made available to other devices and services for communication.
- **Registration Completion:** At the end of a registration, the registration agent sends the newly created device agent ID to the runtime environment messaging application. The application then sends this agent ID to the corresponding devices.

```
{
       "profileIdentifier": "http://iolite.de#TemperatureSensor",
       "deviceAPI": "195187757",
       "name": "TempSensor1",
       "identifier": "insideTemp.jar/TempSensor1",
       "manufacturer":"IOLITE GmbH",
       "properties": [
        {
           "key": "http://iolite.de#InsideEnvironmentTemperature",
           "deviceID": "1729820428",
        },{...},{...}],
}
```

Figure 5: Device Registration Data

2.2 Device Data Update

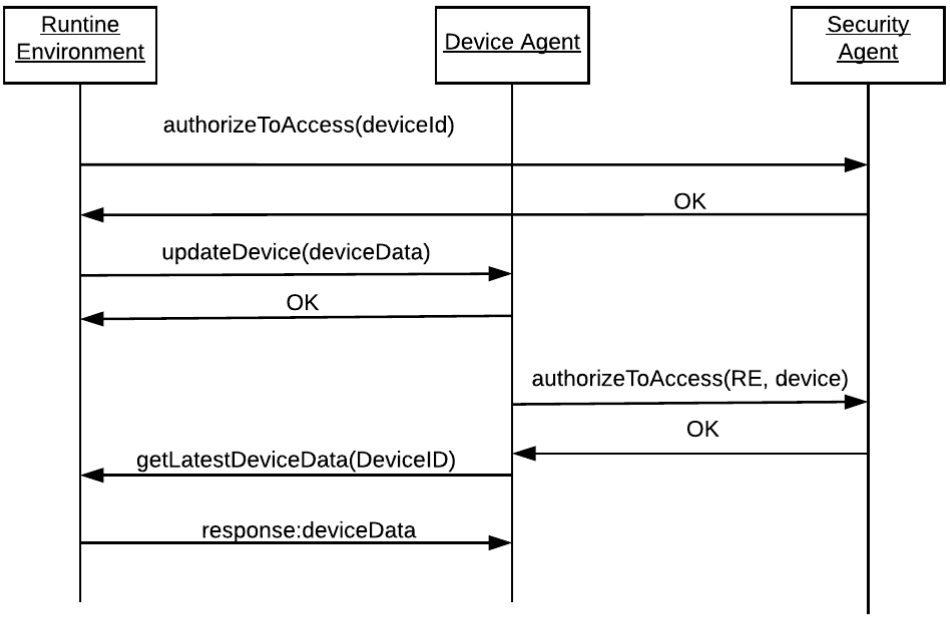

Figure 6: Sequence diagram indicating the device data update process

In the CHARIOT architecture, an RE can also be considered as an IoT gateway that collects all the current device data and forwards these data to the agents in the service layer. Hence,

a communication channel between a device agent and the runtime environment is needed in order to synchronize device data with the corresponding device agents, and a RESTful API communication interface is used for sending data updates to service layer through this channel. A device agent uses an internal RESTful API server to listen for device update messages. The sequence diagram of device data update between CHARIOT components can be seen in Figure 6. The actual data will be sent to the device's agent at the service layer in JSON format. An example message is shown in Figure 7. The actual sensor data does not need to be stored in the directory service, as JIAC agents have their own directory to store data. If another device's agent, or a service asks for the actual data of the device, then they can reach this value by asking the device agent.

```
{
	"Type": "Device Data Update",
	"Content": {
		"ObjectType": "Sensor",
		"ID": 1234567,
		"Data": {
			"Timestamp": "",
			"Value": 22.122
		}
	}
}
```

Figure 7: Device Data Update Message

This architecture saves the directory from congestion arising from unnecessary overhead. Only data updates that need to be sent to the directory service are the ones which change a property of the data profile of a device. For example, if the device has a new functionality, then this information should be added to its device description in the directory service.

2.3 Device Data Request and Response

In case another service or a device agent asks for the device data, this device data request can be forwarded to the device over the device agent. First, a service agent that wants the device data searches for the service description of the device agent in the directory service. After receiving the service description, it communicates with the device agent by using ActiveMQ interface and sends its request. The device agent is responsible for finding the runtime environment to which the device is connected and deliver this request over RESTful API. The runtime environment has a messaging application with a RESTful API server implemented inside. When this server receives the request, it finds the corresponding device and sends the device data update message with the actual data value and the timestamp information back to the device agent. The sequence diagram that explains the data request and response process is given in Figure 8. The response message has the same structure with the device data update message seen in Figure 7. Additionally, a device agent can also define an advertisement interval for its device to adjust the frequency of updates and keep the updated values in its own directory.

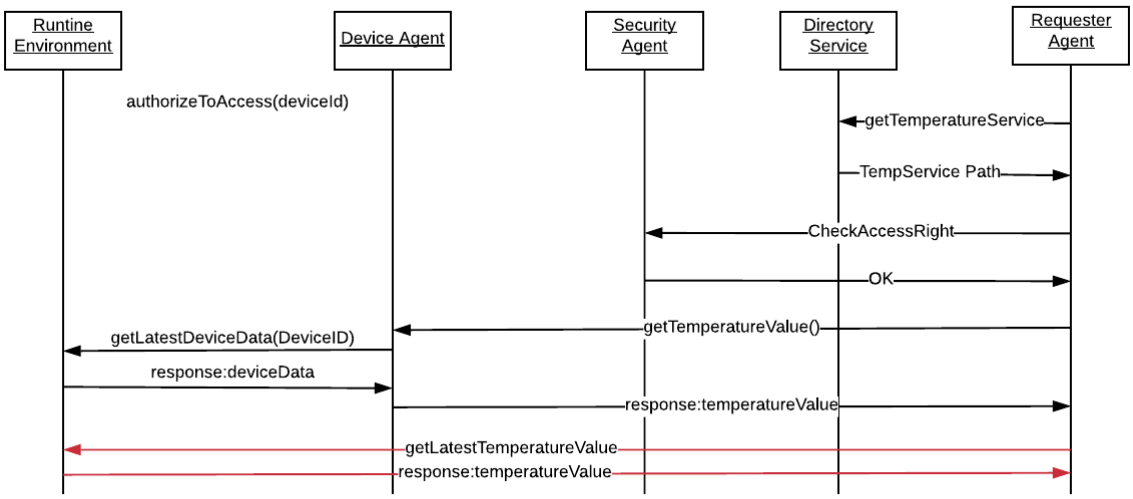

Figure 8: Sequence diagram indicating device data request and response process

Another request message from the service layer that needs to be sent to the device layer is the “heartbeat” message. With an extension to RESTful API protocol, it can be possible not just to register a service but also to keep a heartbeat rate to check whether this device is still active or not. The server can store this heartbeat rate and send periodical messages to check the on/off value of the device. This message uses the same communication channel with the device data request and the response sent to the device agent has the same JSON format. If an off value is received, the device agent can send a new message to activate the device to keep it alive.

2.4 Device Removal

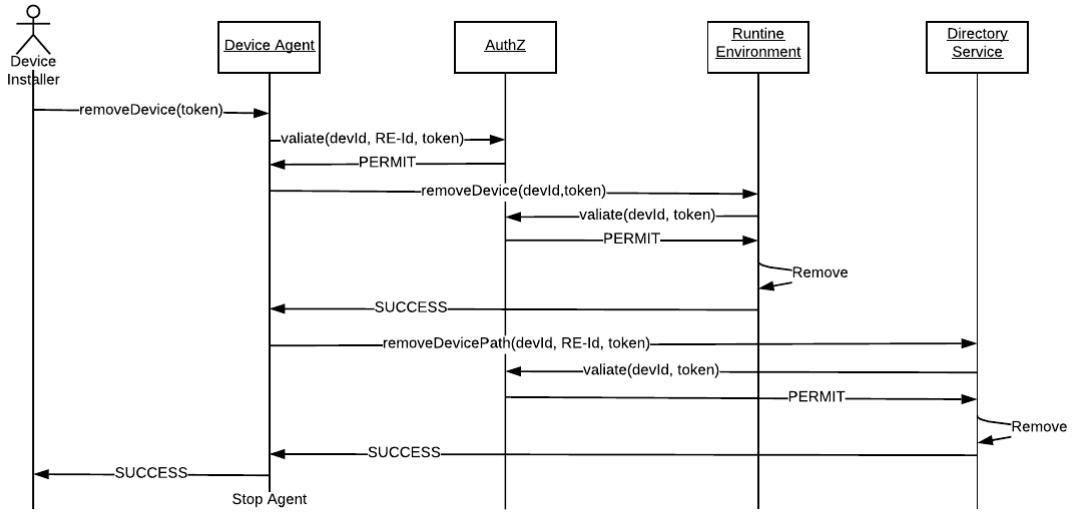

Figure 9: Sequence diagram indicating the device removal process

If a device needs to be removed from the CHARIOT middleware, then a messaging is required between the runtime environment, the device agent, and the directory service. The runtime environment has the option to deactivate a device, and device agent has the right to trigger

this process if it decides to detach the device from CHARIOT by sending device removal message to runtime environment. After completing deactivation, runtime environment sends a response to device agent. The device agent should forward this message to the directory service so that the device description is removed. Device removal sequence diagram is displayed in Figure 9.

2.5 P2P Communication

We assume that device A and device B need P2P communication. We have defined three use cases for this P2P communication:

- **Use Case A:** Device agent A wants to communicate with device B to receive/write some information.
  - ➡️ Agent-to-agent communication, read/write permission
- **Use Case B:** Device agent B wants to communicate with device B to change a value
  - ➡️ Agent-to-device communication, read/write permission, secure channel between device agent and the device
- **Use Case C:** Device A and Device B need to exchange data to complete a process (e.g. production, storage)
  - ➡️ Device-to-device communication, read/write permissions

Use Case A

Device agent A requires current values from one or more devices connected to the CHARIOT middleware. In this case, the device agent A sends a request to the directory service, using RESTful API protocol. Upon receiving the request, the semantic search component (SeMa) searches the directory service to find a matching device description. After making a search in the DS, SeMa returns the set of matching descriptions in the form of a ranking list (a list of service ordered by required criteria). After receiving the result, the device agent A may choose to send a communication request to the device agent on top of the list, or to a group of device agents. Let us assume that device agent B is on top of the list. The description of device agent B is given to device A only if the search matches the profile of device B and if device agent A has the permission to talk to device B. After obtaining device description of B, device agent A uses ActiveMQ protocol to communicate with device agent B. As the actual value of the sensor data is stored in the directory of device agent B (check device data update section), device agent B does not need to send a request to device B, meaning that this communication happens only in the service layer. Device agent A can also ask for historical data of device agent B, if the the device agent B stores the data updates provided by device B. The details of use case A is presented in Figure 10 with a sequence diagram.

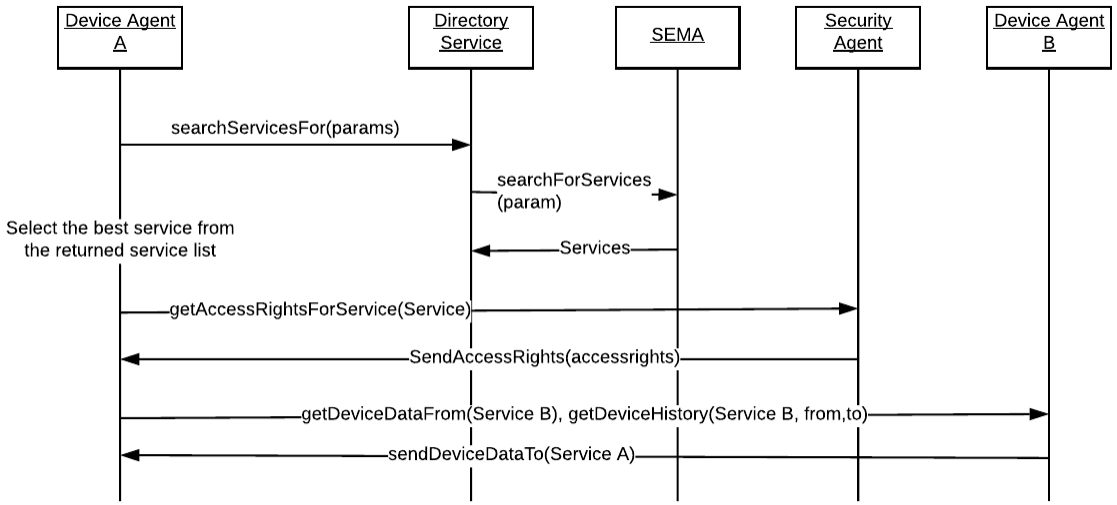

Figure 10: Sequence diagram indicating Agent-to-Agent Communication

Message Format: Session Token | Service A access rights for Service B | Payload

Use Case B

In the second use case, a similar structure is used until establishing communication between device agent A and device agent B.

But in this use case, device agent wants to change a value in device B; therefore, the device-to-device communication should include service layer to device layer communication as well. Service layer to device layer communication happens through the message bus defined between the device agent B and the messaging application of the runtime environment to which device B is connected. RESTful API is used for this communication. Once the message reaches the runtime environment, the communication from runtime environment (RE) to the device is handled internally by the RE. An important task in this use case is to ensure a secured communication channel (message bus) between runtime environment and CHARIOT middleware with SSL/TLS or another secure method. Device Agent A also needs to have write permission in order to change a value of Device B. The details of the messaging between CHARIOT components in use case B is presented in Figure 11 with a sequence diagram.

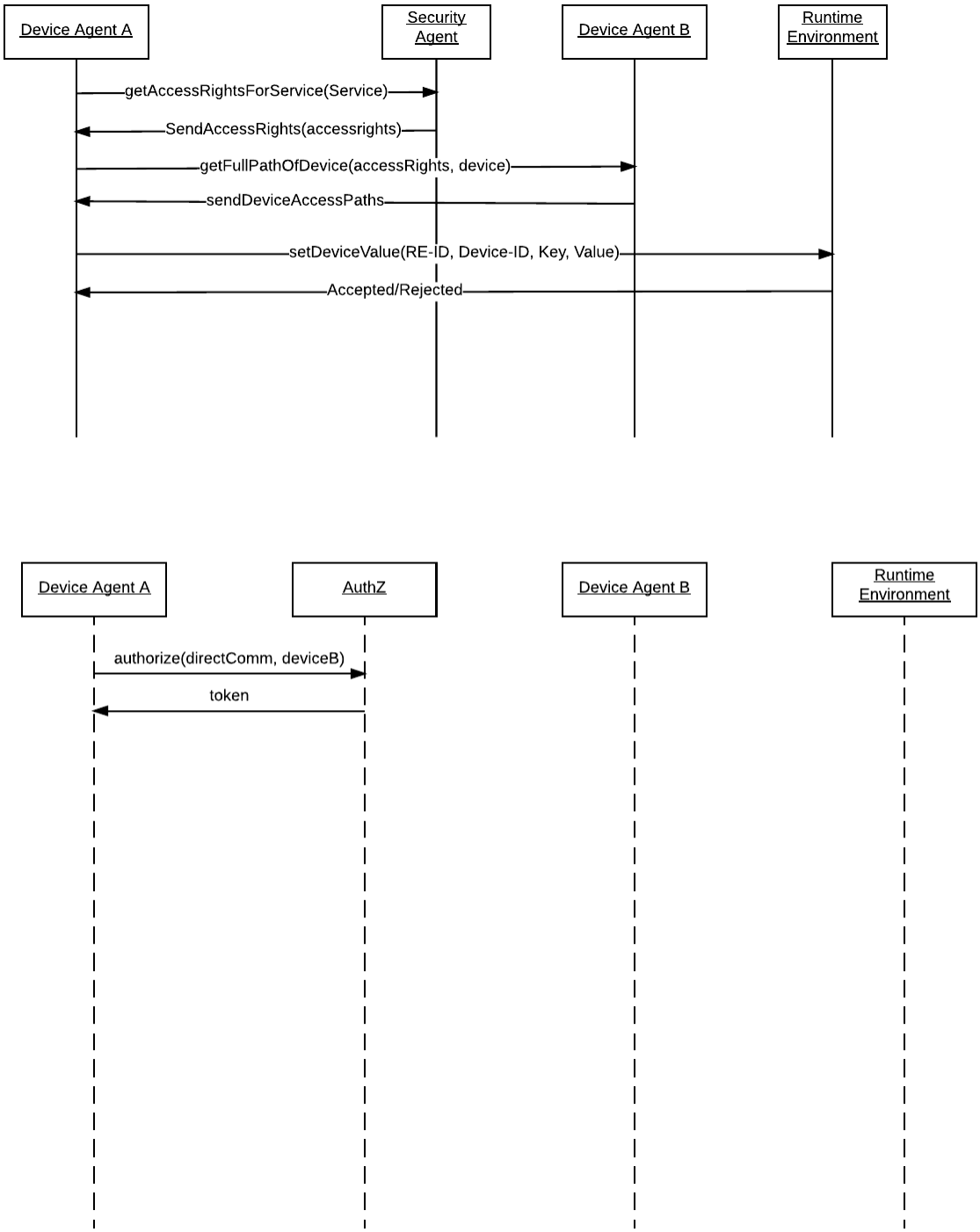

Figure 11: Agent-to-Device Communication

Message Format: Session Token | Service A access rights for Service B | Service A access rights for Devices of B | Payload

Use Case C

Communication layer is also responsible for data exchange between the devices in the device layer through the IoT message bus that connects the runtime environments. This sort of

communication also begins at the service layer and data exchange in the device layer begins only after device agents at the service layer agree on P2P communication. The device agents inform their devices to start the communication at the device layer. For device-to-device communication, a new IoT message bus have to be designed to establish communication between the runtime environments. RESTful API communication interface is to create a common message format for the interaction between devices running on different runtime environments. The messaging application in the runtime environment is used to collect the registered device data, and the message bus between the runtime environments is used for data exchange.

Another task in this use case is to transfer of some service functionalities to the device layer for P2P communication. In that case, RE that serves as an IoT gateway handles P2P communication to decrease the latency and to enable data processing at the edge without transmitting the data to the service layer. Executing service tasks on the runtime environment requires transferring functionalities over RESTful interface between the device agent and the runtime environment. This feature would be an optional extension to CHARIOT middleware, and a decision on its deployment depends on the time constraints within the project. The details of the messaging between CHARIOT components in use case C is presented in Figure 12 with a sequence diagram.

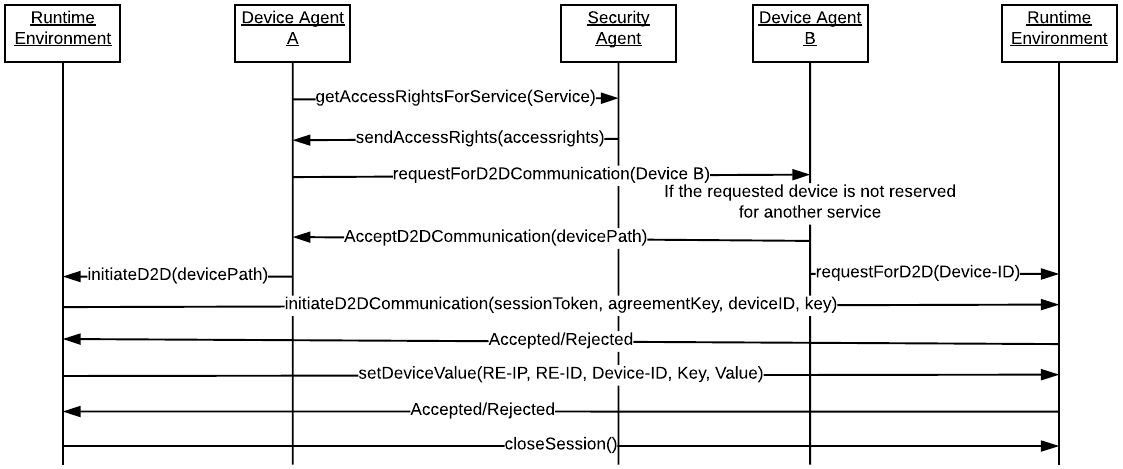

Figure 12: Device-to-Device Communication

2.6 Other Message Types

If a message has a type that is not defined in this section, then this message should be acknowledged as Unknown by the communication components such as the runtime environment messaging application, device agent, registrar agent, and the directory service. If a message contains an information that is not valid, (e.g. an on/off value is sent instead of a temperature value), then an error message must be sent to the transmitter in response. The unknown and error message types defined in the communication interfaces (i.e ActiveMQ, RESTful API) will be used in CHARIOT middleware.

3. Distributed and Scalable Directory Service Architecture

The main parts of the architecture of the scalable and distributed service are highlighted below, as it also hints at the roadmap for the ongoing and future work in AP3. The most suitable architecture that meets the demands of an Industry 4.0 scenario is a decentralized DS architecture as a centralized directory service is not preferred in an IoT scenario, due to the rapidly increasing number of devices leading to increasing overhead, delay and other scalability problems in general.

The directory service in CHARIOT relies on ICN technology [1] for a scalable and distributed solution. An ICN-based solution helps us to use already implemented network solutions for IoT related problems such as name-based routing to avoid congestion, keeping delay and message overhead at minimum using built-in caching mechanism. The ICN-based Directory Service architecture envisioned for CHARIOT is given in Figure 13.

Service and devices are semantically described using OWL-S framework. This allows other components, i.e., planner, to exchange and derive knowledge about processes using ontology reasoning. The huge number of OWL-S descriptions are distributed among Directory Service Nodes (DS-Nodes) with optimized replication strategies to achieve availability, reliability and responsive queries. Name-based approach allows queries to the directory service to be formulated with service attributes using a specially designed ICN naming scheme. Upon receiving a query, DS-Node will be able to search through the OWL-S descriptions with the help of service matcher component SeMa [2], developed under JIAC project.

3.1. Directory Service Architecture

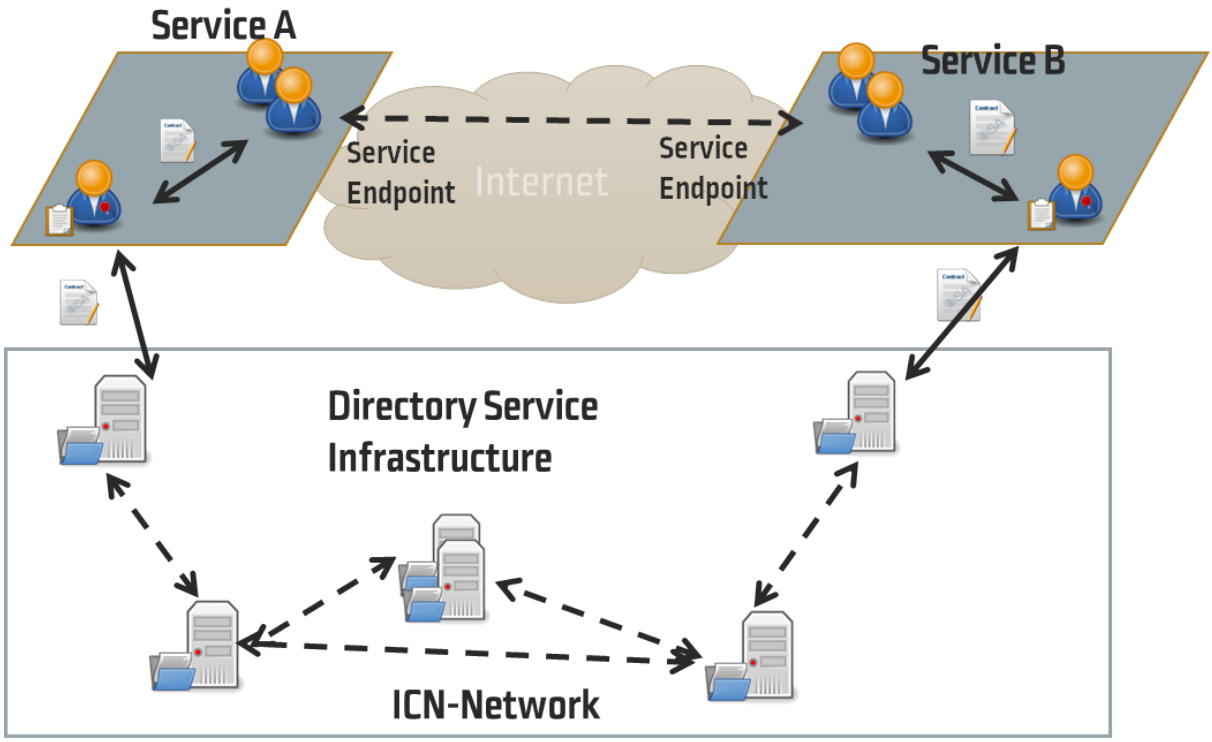

Figure 13: Directory Service Architecture Diagram

The main purpose of Directory Service (DS) is to provide a central knowledge base for storing a description of services, IoT devices, their behavior and interactions within various business processes. However, one single directory entity cannot meet the scale and distributed nature of IoT applications. The DS architecture is therefore designed as a mesh of distributed data storage nodes, each of which maintains a unified, consistent data retrieval and query interface. This results in a logically centralized service from the middleware perspective but still has the scalability of a geographically distributed infrastructure. The challenges for the realization of such design are data distribution between DS-Nodes, query latency, and security, among others. Additionally, CHARIOT middleware is realized with JIAC MAS framework. Another important part of the architecture is, therefore, an agent interface, which realizes the data management protocols and enables the integration of DS and CHARIOT middleware.

3.1.1 Distributed Directory Service

To meet the latency requirements and to provide increased performance for the platform, a distributed directory service design is favored. In most current widely used distributed system, the topology of distributed nodes closely resembles the organization of data, e.g., hierarchical tree, interconnected domains. This results in a static design that requires the system to adapt to information models. DNS is a hierarchical directory example in which the hierarchy is formed based on the geographical location of stored entities (e.g., domain names). However, IoT services consist of a large number of cross-domain applications, devices, which inter-connected in a dynamic and unforeseeable patterns. This makes restriction or organization of service data according to geographical or domain specific structure non-optimal. In contrast, we take a data-centric approach to the design of DS infrastructure. The design does not rely on any specific topology. Instead, the DS-Nodes can be added or removed according to IoT service and application design, resulting in a self-organized network for data distribution to meet the changing demand for latency, reliability, etc. For this purpose, each node has a uniform architecture and is connected to other nodes with a data-centric communication protocol. Details of the data-centric network and DS-Node architecture are given in later sections.

Being independent of DS topology, different strategies for data distribution among DS-Nodes can be tailored for specific service model and query patterns, e.g., when a hierarchical data model is applied, each node can store a subtree of the whole data model, or a complete tree with reduced number of service descriptions at each level. This enables the balanced distribution of data throughout DS infrastructure.

3.1.2 Scaling Directory Service

The independence on physical topology results in a flexible DS that scales itself to meet service demand. A statically designed infrastructure must take future usage and demand into account, which often leads to poor resource allocation and balancing. In such a system, over-provisioning is common to cope with unpredicted demand, which requires complex offline approaches to reduce redundancy and maintain consistency. Data-centric DS infrastructure relies on real-time approaches, which are adaptive to system load, e.g., content caching, request routing. Efficient caching and routing algorithms aim at reducing query delay by

placing relevant content on nearby DS-Node, and number queries and data traffic within the DS infrastructure.

In order to achieve scalability in the service level, the directory service is implemented outside the agent platform. This way, scaling the directory is independent of the constraints of the single JIAC agent. In the current JIAC architecture, each directory and the node has to store all actions and update through discovery which requires broadcasting, and hence there is a scalability problem. While agents represent IoT services and applications, DS may also serve as a service broker, which provides agent descriptions and facilitate their interaction.

Due to the uniformity of DS-Node architecture, Local demand surge in a specific location is addressed by dynamic scaling out DS-Node. Additional numbers of DS-Nodes can be provisioned on demand at the location without the need for manual setup and reconfiguration of the DS network. This enables DS to take advantages of Cloud Computing to handle scalability of DS resources, e.g., a cloud management system can provision a virtual machine that increases storage and processing capabilities and deploy a DS-Node, thereby increasing the scalability of the system.

3.2. NDN-based DS Infrastructure

As previously mentioned, data-centric approaches are fundamental to enable scalability, adaptability, and network topology independence of DS infrastructure. Named Data Networking [3] (NDN, also known as ICN) is such a data-centric networking approach for the exchange of data between DS-Nodes. The underlying principle is that a communication network should allow a user to focus on the data he or she needs, named content, rather than having to reference a specific, physical location where that data is to be retrieved from, named hosts. Named-data networking comes with the potential for a wide range of benefits such as content caching to reduce congestion and improve delivery speed, simpler configuration of network devices, and building security into the network at the data level. Communication in NDN is driven by receivers, i.e., data consumers, through the exchange of two types of packets: Interest (INT) and Data (DATA). Both types of packets carry a name that identifies a piece of data that can be transmitted in one Data packet. The consumer sends interest packet with description (name) of the data it needs. When an intermediate node receives the interest, it looks for the data in its content store (CS). If the data is not found, it forwards the interest to the next nodes and keeps track of the incoming and outgoing interfaces for this data in pending interest table (PIT). A series such forwarding actions create a breadcrumb path the INT has passed. When the interest arrives at the source node, the requested data is put in a data packet (DATA) and sent back the path towards the consumer. A previous forwarding node receives the DATA, it removes the data name entry in PIT table and adds an entry with the name and interfaces to forward the data to consumers in the FIT table. It also put the data in CS for subsequent INT.

In CHARIOT DS infrastructure, the data to be exchanged are services and devices descriptions, which mainly contain various attributes. Using NDN enables the DS to decouple the data from locations of the nodes that store the data while taking advantages of data forwarding and caching mechanism [1]. A naming scheme enables expression of service attributes that can be used to look up the services regardless of where they are stored. E.g., “Icn://de.dailab.iot/sensor?lat=35,lon=11,radius=1km,scale=census” contains a rich semantic

describing a sensor's domain, location, type, etc. Various caching and forwarding strategies can be designed to best serve the demand and DS infrastructure performance.

The directory service relies on ICN technology for a scalable and distributed solution. An ICN-based solution helps us to use already implemented network functions for IoT related problems such as name-based routing to avoid congestion, keeping delay and message overhead at a minimum, using a built-in caching mechanism. Our idea is to directly deploy ICN functions that can specifically handle such problems. For example, developing content-based search and content caching strategies for ICN to be used in DS structure.

3.3. Architecture of DS-Node

DS-Node is designed as a uniformed, standalone DS, which can be connected with additional DS-Nodes to form a larger, distributed infrastructure. Additionally, it supports the extension of basic functionalities with forwarding and caching strategies implementations without the need for reimplementation, redeployment, enabling extendability, backward compatibility and portability of the infrastructure. Karaf platform [4] is therefore selected as the software platform for DS-Node implementation, which supports both standalone and cloud-based deployment.

The architecture diagram of the directory service is given in Figure 14, and it contains following components:

- RESTful API: provides a universal communication interface to communicate with JIAC agents to query and retrieve service descriptions in OWL-S format.
- Web UI: is the user interface designed for service providers for describing and managing provided services and devices.
- Model: Service and device data model based on service ontology.
- Directory Service: The logic layer defining data management, adaptation, access control, etc
- Ontology database: Datastore for semantically annotated service descriptions.

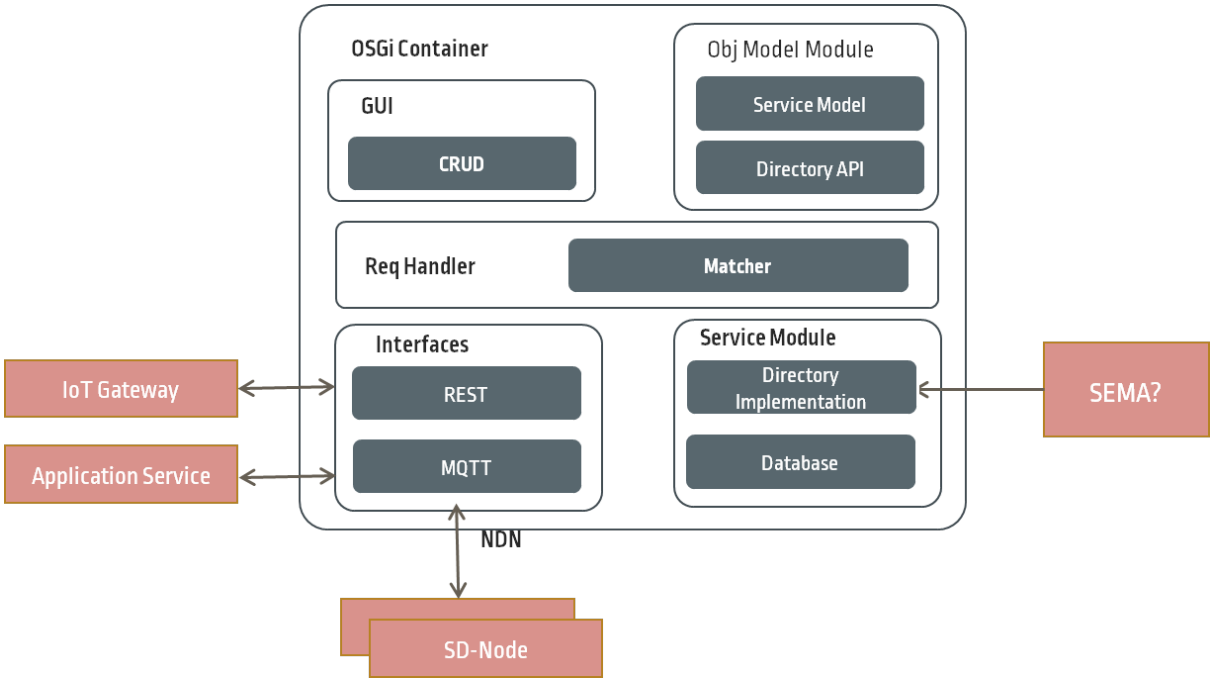

Figure 14 - Directory Service Implementation in CHARIOT

3.3.1 GUI

DS-Node provides an user interface allowing description of services and devices to be provided and managed by users (service providers, device providers). The descriptions are directly stored on the DS-Node. It becomes the data source of the descriptions. NDN query forwarding mechanism allow queries for the description received by other DS-Node to be routed to this DS-Node.

3.3.2 Interfaces

Interactions between the DS-Nodes and other components, which provide and consume the stored service description, are realized through different protocol specific interfaces.

1. CoAP/MQTT Interface

This interface implements the protocols defined in Section 2 for IoT domain. The protocol messages, e.g., registration, update, etc., are transported with constrained M2M communication protocols, i.e., CoAP, MQTT. Respective CoAP or MQTT server is implemented in DS-Node allowing it to receive service descriptions and queries from IoT Gateways or devices.

2. REST Interface

Similar to M2M Interface, a Rest server is implemented to allow external service component to make queries over the Internet using the HTTP based REST protocol. Software agent and web services are often deployed on less constrained platform can interface with DS infrastructure over the Internet.

3.3.3 Directory Implementation

This component implements business logic of a DS-Node. It handles different types of queries by interacting with other components in the DS-Node. For example, once a request for service description is received from Rest interface, the implemented logic triggers matching component and search local database. If the description is not found locally, queries forwarding mechanism is triggered allowing the query to be forwarded through NDN overlay to other DS-Node. If the description is found in other DS-Node, it is returned to the requester and cached by this node.

3.3.4 Ontology Database

An ontology database is required to construct a common URI-like OWL-S format to store the names, capabilities and communication related paths, addresses and protocols of the devices and services. Device descriptions are prepared based on device models and these models are stored in a library in the communication layer as part of the DS structure. The models are based on the unified device models used inside the runtime environment.

Use of ontology provides structured data to machines to automatically understand the content of the services. The device description in the DS should enable semantic searching of devices by the services, thus the URI-like addresses of devices must support semantic searching. For this reason, the DS must be able to store service descriptions written in OWL-S ontology.

Apache Jena, Apache Fuseki and SPARQL are the components used for the ontology database of the directory service. Apache Jena is a framework for building semantic web applications [5]. It contains API's for Resource Description Framework (RDF) and OWL. The Resource Description Framework (RDF) is a standard for describing resources, which are identified by URIs. Apache Fuseki provides REST-style interaction with the RDF data [6]. Fuseki war is deployed in the Apache-tomcat. After deploying the war file, Fuseki provides endpoints to insert, update, read and delete triples via HTTP. It also provides User Interface to query in Web console. Apache Jena API is used to form the Query and call the HTTP endpoints. SPARQL [7] is a query language that is considered for CHARIOT, as it can be used to express queries across diverse data sources, and contains capabilities for querying required and optional graph patterns. Although the SPARQL queries can be sent to Jena database, they only affect content locally available in the DS-Node. The DS-Node interfaces (REST, CoAP) should be used instead as query interfaces, so that the queries can be forwarded and data caching is activated throughout the infrastructure.

4. Semantic Search Component

Semantic search component enables semantic matching of search queries of various services that are connected to CHARIOT platform. The semantic search component should be able to find the fitting device and its functions using semantic matching of services. For this purpose, use of SeMa[2] semantic service matcher, the semantic matcher developed by JIAC team [8] in DAI-Labor, is an attractive option when we improve its functionalities to suit IoT use cases in CHARIOT. SeMa carries out service matching tasks with the help of experts that are responsible for finding structural, logical and semantic relatedness of services. These

analyses return a set of matching services in the form of a ranking list to semantic search queries. SeMa is an implemented JIAC bean that searches for a matching service description; however, as stated in [2], SeMa's core matching algorithms are JIAC independent, which makes it possible to use SeMA in the directory service built outside JIAC. SeMa architecture is based on OWL-S.

Semantic search queries are defined between the JIAC agents representing Industry 4.0 services and the DS. In these search queries, an agent may ask for a device or a group of devices sharing the same properties. When a search query comes to the directory service, semantic search query translator is used to match the service request to the device model to find the queried device and its runtime environment. At the end of this search process, the DS must be able to find the queried device or the group of devices and send the device properties and network configuration information (IP address, protocol to reach the device, etc.) to the agent via the semantic search query translator.

During these queries, the DS should find an efficient method to filter out unrelated DS nodes to decrease the latency of a search, and this filtering is also the task of this translator. These queries are going to be used in a directory service with distributed databases; therefore, scalability and flexibility of the search algorithm plays a key role in terms of latency and narrowing the search space. Semantic technologies are used during this search for filtering purposes as semantic descriptions provide structured data that helps in decreasing the latency of the search. Semantic searching is used in the service layer and finding the matching service means finding the matching device, as all devices in CHARIOT are represented as services.

The first function of semantic search query translator is to define and formulate search queries in an optimal way to query OWL-S descriptions of devices in the directory service. The semantic search query translator is responsible for receiving requests in OWL-S format, as the DS contains OWL-S descriptions of device representing services. In the current version of JIAC, whenever the directory of the JIAC agent receives a "searchAction" request with a search query, it checks whether that query has a semantic URI defined, pointing to the location of the OWL-S description. Then the agent directory delegates the matching process to the SeMa which will get the OWL-S from the URI and use that for matching against other services that also have URIs pointing to their OWL-S descriptions. However, in an IoT use case, we expect many devices to be integrated into the CHARIOT platform, making this solution inefficient. In CHARIOT communication layer, all queries should go to the DS that holds all service descriptions. The search mechanism needs a sophisticated approach to filter candidates that are presented to the SeMa². Another option that is considered is to integrate SeMa² into the DS infrastructure so that the matching algorithm itself can optimize its speed.

The most important functionality of semantic search query translator is to search the request inside the directory. SeMa carries out service matching tasks with the help of experts that are responsible for finding structural, logical and semantic relatedness of services. These analyses return a set of matching services in the form of a ranking list to semantic search queries. In CHARIOT architecture, SeMa is carried to the communication layer as a Java bundle that does semantic search query translation, in order to handle search queries inside the DS. Finally, after the query matches the relevant service that represents the device, this result should be returned to service that queried for the device and forwarded to the RE to which the queried device is connected.

4.1 Simple Search

If an agent needs to reach the instant values of a device, then a simple search option is also provided to the user. This simple search is done by the agent platform with matching functions through actions and existing directories inside a JIAC agent, bypassing the semantic search. Both search for action and search for agent options are possible. These action descriptions are not comprehensive when compared to semantic service descriptions, e.g., no preconditions are defined in an action. There are also various search criteria to filter a request such as provider, etc. For example, *EqualityChecker* function of JIAC agents can be used to check whether a sensor data has reached a threshold in a rule-based time critical process.

5 Conclusion

In this document, we give the details of the communication layer architecture in CHARIOT, by explaining the messaging between CHARIOT components, scalable and distributed directory service, and the semantic search component. We finalized the document after contacting our technology partners in JIAC and IOLITE, and arranging meetings with them to understand the specific requirements of these technologies and what they can provide to support the communication infrastructure of CHARIOT middleware.

Message formats define the process of integrating a device to CHARIOT, and how this device can be used by other components inside the CHARIOT platform. The runtime environment provides device models and the necessary abstraction to deal with the heterogeneity, and the semantic descriptions make use of this abstraction. The solution in CHARIOT moves the second directory outside the JIAC agent platform for scalability and a distributed architecture. Thus, additional message formats are defined between the directory service and the agents. ICN-based network functionalities are to be used to manage the directory service efficiently. Finally, we have defined the semantic search component that enables other services to make use of devices and services in CHARIOT.

References

[1] I. Abdullahi, S. Arif, and S. Hassan, “Survey on caching approaches in information centric networking,” J. Netw. Comput. Appl., 2015.
[2] Fähndrich et al., “Semantic Service Management and Orchestration for Adaptive and Evolving Processes”
[3] https://named-data.net/
[4] https://karaf.apache.org/
[5] https://jena.apache.org/getting_started/index.html
[6] https://jena.apache.org/documentation/serving_data/
[7] https://www.w3.org/TR/rdf-sparql-query/#termConstraint
[8] http://repositories.dai-labor.de/sites/jiactng/5.2.2/