Smart Systems Service Infrastructure (S^3I)

Design and deployment of the Smart Systems Service Infrastructure (S^3I) for decentralized networking in Forestry 4.0

A KWH4.0 Position Paper

– English version –

31.01.2022

Kompetenzzentrum Wald und Holz 4.0 | Center of Excellence for Forestry 4.0
c/o RIF Institute for Research and Transfer e.V.
Joseph-von-Fraunhofer-Straße 20
44227 Dortmund, Germany
www.kwh40.de

Contact

Kompetenzzentrum Wald und Holz 4.0 | Center of Excellence for Forestry 4.0
c/o RIF Institute for Research and Transfer e.V.
Joseph-von-Fraunhofer-Straße 20
44227 Dortmund, Germany
www.kwh40.de

Contact person: Dipl.-Ing. Frank Heinze
Phone +49 (0) 231 9700-781
frank.heinze@rt.rif-ev.de
Responsible author: Dr. Martin Hoppen, MMI

Authors

RIF Institute for Research and Transfer e.V.
General Secretary: Dr. Svenja Rebsch, Dipl.-Inf. Michael Saal
Joseph-von-Fraunhofer-Straße 20, 44227 Dortmund, Germany

Laboratory for Machine Tools and Production Engineering (WZL)
RWTH Aachen University
Director: Prof. Dr.-Ing. Christian Brecher
Steinbachstraße 19, 52074 Aachen, Germany

Institute for Man-Machine Interaction (MMI)
RWTH Aachen University
Director: Prof. Dr.-Ing. Jürgen Roßmann
Ahornstraße 55, 52074 Aachen, Germany

Institute of Industrial Engineering and Ergonomics (IAW)
RWTH Aachen University
Director: Prof. Dr.-Ing. Verena Nitsch
Eilfschornsteinstraße 18, 52062 Aachen, Germany

Landesbetrieb Wald und Holz Nordrhein-Westfalen

State Enterprise Forestry and Timber NRW
Center for Forestry and Timber Industry
Forestry Education Center
Director: FD Thilo Wagner
Alter Holzweg 93, 59755 Arnsberg, Germany

Funding

This project is supported by the state of North Rhine-Westphalia using funds from the European Regional Development Fund (ERDF).

EFRE.NRW Investitionen in Wachstum und Beschäftigung

EUROPEAN UNION
Investing in our Future
European Regional Development Fund

Together with the project “iWald”, supported by: Federal Ministry of Food and Agriculture by decision of the German Bundestag (Funding code 22012718).

Federal Ministry of Food and Agriculture

FNR Fachagentur Nachwachsende Rohstoffe e.V.

Version	Date	Sections	Changes
1.0	19.12.2019	All	First version (German)
2.0	28.02.2020	1.4, 2.3+4, 4	S^3I Config, S^3I Repository, S^3I-B Messages (German)
2.0.1	31.01.2022	All	First English version

Smart Systems Service Infrastructure (S³I)

The basis of communication in Forestry 4.0 is decentralized networking in an Internet of Things (IoT). These ***F4.0 Things*** comprise ***F4.0 Components*** (assets such as machines, devices, people and their ***Digital Twins***), ***F4.0 Services*** (software services), and ***F4.0 Human-Machine Interfaces*** (***F4.0 HMI***) (desktop/web applications, apps ...), which are networked together situation-specifically to form ***F4.0 Systems*** (see Figure 0-1) in order to map ***value chains*** and, thus, value creation processes. These F4.0 Things can be mapped by various technological approaches in the process. Digital twins can either have their runtime environment directly at or on their asset (***edge approach***) or be operated on a server (***cloud approach***); there are mixed forms in between (***fog approaches***). The approaches have different advantages and disadvantages, depending on the asset and the deployment scenario. Forestry machines such as harvesters benefit from the edge approach because data is processed locally and then transmitted via radio links. Assets that are not machines (forest, log pile etc.), on the other hand, require a runtime environment (hardware and software) in the cloud. The same considerations can be made for services (local localization service in the edge approach versus forest growth simulation service in the cloud) or human-machine interfaces (local app on a smartphone versus web app in the cloud).

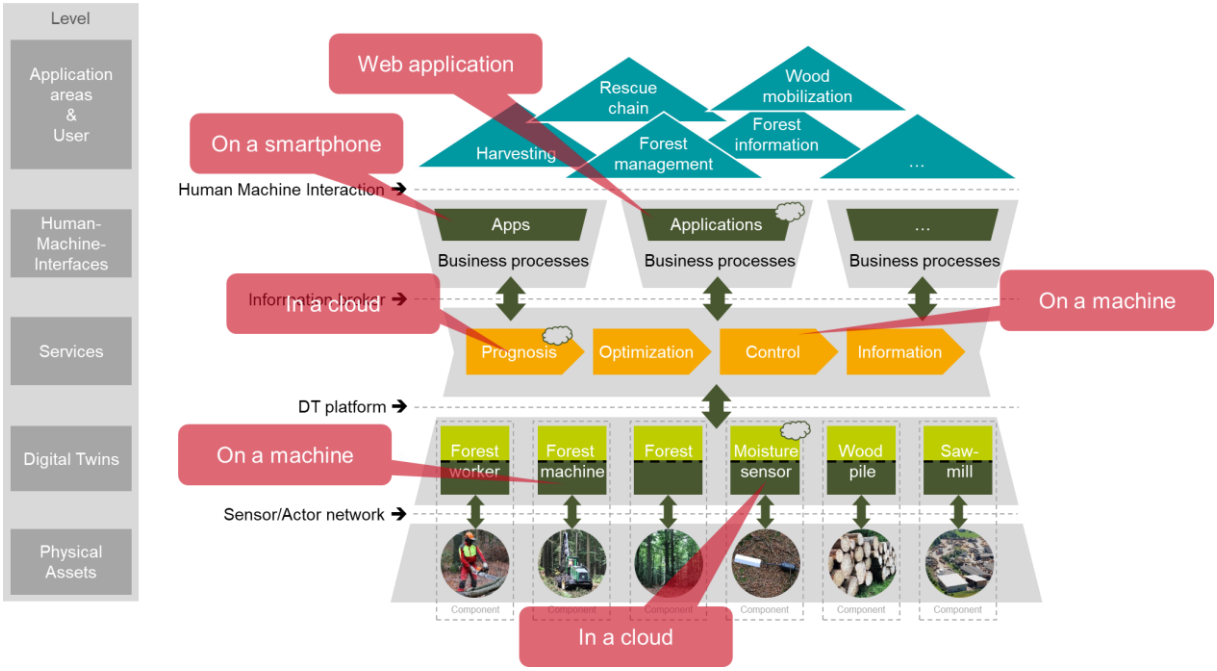

Figure 1: Networking of "things" in Forestry 4.0 and their location on the machine (edge) or in the cloud[1]

In addition to this technically motivated heterogeneity, there is also a motivation at the organizational level in the forest and wood sector. Different actors typically do not want to run their F4.0 Things on a central platform, but make their own choices.

In order for networking and, thus, communication to be possible in this decentralized IoT (or Internet of F4.0 Things), a central infrastructure is needed consisting of a few central software services (these are not F4.0 Services themselves), via which F4.0 Things can authenticate themselves (as far as possible

[1] Derived from Bosch Software Innovations 2012. Photos (v. l.): A. Böhm, RIF; F. Heinze, RIF; S. Wein, WZL; A. Böhm; S. Wein; Michael Lorenzet / pixelio

only once), find each other and communicate with each other – considering the authorizations assigned to them in each case. KWH4.0 is developing the ***Smart Systems Service Infrastructure (S³I)*** for these purposes and making it available to all interested partners.

In the following Section 1, the basic concepts of S³I are explained. Section 2 then goes into detail about the data models in use. Section 3 describes the authorization in the directory, Section 4 describes the encryption concept for messages, Section 5 the authentication and Section 6 shows use cases for the use of S³I. This position paper is also supplemented by various programming examples, which can be found at https://git.rwth-aachen.de/kwh40/s3i.

1 S³I Concept

Figure 2 shows the basic concept of S³I.

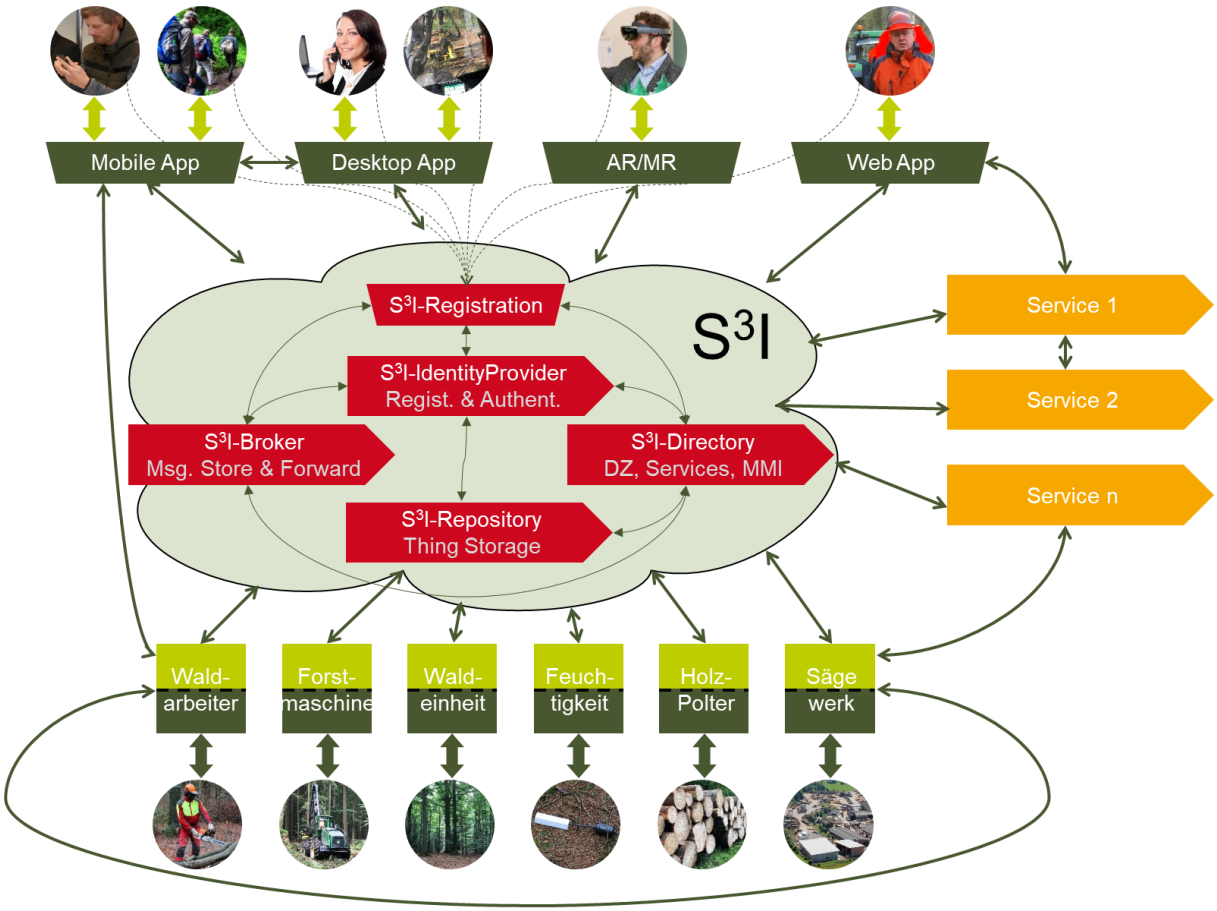

Figure 2: Basic concept of the Smart Systems Service Infrastructure[2]

1.1 S³I Directory

S³I Directory is a core element of S³I and represents a directory service, in which information about all F4.0 Things is managed. Here, they can be registered, provided with information and modified, searched for via various properties and deleted again, if required. Authentication (in response to the questions "Who are you?" and "Are you really who you say you are?") is the task of the ***S³I IdentityProvider*** (see below). Access to the entries in the directory is authorized on a role basis via policies (which considers the question "What are you allowed to do?"). Here, it is possible to specify in detail, down

[2] Fotos: (o.) A. Böhm, RIF; Rainer Sturm / pixelio; Konstantin Gastmann / pixelio; Stefan Wein, WZL; A. Böhm; Peter Kamp / pixelio; (u.) A. Böhm; F. Heinze, RIF; S. Wein; A. Böhm; S. Wein; Michael Lorenzet / pixelio

to the level of individual entries, who is allowed read, write, or delete access to which information (see Section 3)

The information about the F4.0 Things in the S³I Directory includes general information about their identification, type, roles, essential attributes, (often) spatial location, and the data model in use. A central component is also information on the accessibility of the respective F4.0 Thing in the form of so-called endpoints via protocols such as OPC UA, MQTT, REST or via the message-based ***S³I-B protocol*** provided by ***S³I Broker*** (see below). The exact structure of the directory is described in Section 2 of this position paper. Authentication to S³I Directory is done via JSON Web Tokens (JWT)[3] provided by ***S³I IdentityProvider***.

S³I Directory is implemented using Eclipse Ditto[4], an open source platform for Digital Twins with MongoDB backend and a policy infrastructure for the management of authorization data, which is also used in industry.

1.2 S³I IdentityProvider

S³I IdentityProvider represents the central service for registering and managing the identities of both people themselves and F4.0 Things. It enables authentication ("It's me!"), provides authentication ("Is it really him?"), and allows for single sign-on (SSO). For this purpose, it manages user names, passwords (for people) or a secret (for F4.0 Things) for each identity. After logging in once to S³I IdentityProvider with user name and password or with secret, it issues a JSON Web Token for authentication to others. This JWT can be used to authenticate to S³I services (directory and broker) as well as to any F4.0 Things (if they can and want to support it). Overall, this enables SSO across the F4.0 IoT.

S³I IdentityProvider is implemented using the open source software Keycloak[5], which is used in industry, as well.

1.3 S³I Broker

S³I Broker provides all F4.0 Things with an S³I-integrated service for message-based communication, comparable to a formalized e-mail. It represents an optional building block of S³I that can, but does not have to, be used by an F4.0 Thing for networking in addition to other transport protocols such as OPC UA, MQTT or REST. The advantage of the integrated approach is that things can communicate with each other with relatively little effort. By using formalized messages, the ***S³I-B protocol*** is realized. In addition to the formalization of the structure (see Section 2.3), it also includes an integrated approach to asymmetric encryption, which is implemented using public keys stored in S³I Directory. Messages can be used in all directions between all F4.0 Things, such as between humans (more precisely: their HMI), between humans and machines (DT or service), and between machines and between services. The store-and-forward approach implemented in this process solves the problem of frequent non-accessibility of F4.0 Things in the IoT of Forestry 4.0. Other classic Industry 4.0 or IoT protocols are less suitable for this purpose because they work connection-oriented and, therefore, require a permanent communication link.

The technical basis of the S³I Broker is the Advanced Message Queuing Protocol (AMQP)[6]. In interaction with an AMQP broker, any messages can initially be exchanged asynchronously, stored in so-called queues for the addressed F4.0 Thing.

[3] https://tools.ietf.org/html/rfc7519
[4] https://www.eclipse.org/ditto/
[5] https://www.keycloak.org/
[6] https://www.amqp.org/

The technical implementation of the S³I broker is based on the open source AMQP broker RabbitMQ[7], which is also used in industry.

1.4 S³I Registration

While the aforementioned services themselves provide only machine-grade interfaces, the ***S³I Registration*** web application allows people to access S³I to manage (create, modify, delete)

- their own identity as well as identities of F4.0 Things they manage, and
- these F4.0 Things themselves

In the background, S³I Registration accesses the individual S³I services.

The technical implementation of this web application is currently in internal review.

A corresponding ***S³I Config*** service is already available. Via a REST API, it enables the registration and deletion of persons as well as the creation and deletion of F4.0 Things. ***S³I Config*** ensures that these actions are performed in a coordinated manner in all three sub-services (IdentityProvider, Directory and Broker) of S³I.

1.5 S³I Repository

As a supplement, S³I offers ***S³I Repository*** as a cloud storage for F4.0 Things. In particular, Digital Twins can be hosted in this (as well as S³I Broker) optional additional offering.

As with S³I Directory, the technical basis for S³I Repository is Eclipse Ditto. Unlike the directory, however, not only the "phone book entries" for F4.0 Things are managed here, but the F4.0 Things' data itself is stored. Accordingly, it is an independent Ditto instance, in addition to the one for S³I Directory.

2 S³I Data Models

This section defines conceptual data models used in S³I Directory, in S³I IdentityProvider as well as in the S³I-B protocol as well as their mapping to JSON structures.

2.1 S³I IdentityProvider

The left part of Figure 3 shows the ***conceptual data model of S³I IdentityProvider***. It comprises the management of identities that can be identified via an *identifier.* A distinction is made between identities describing F4.0 Things (*ThingIdentity*) and those describing persons using S³I (*PersonIdentity).* As described above, persons use a user name (*username*) and password (*password)*[8] to log in, while F4.0 Things use a *secret.*

There are links with the data model of S³I Directory. Each F4.0 Thing in S³I Directory is assigned exactly one *ThingIdentity* in S³I IdentityProvider (via *thingIdentity*), via which it is identified. In addition, persons (*PersonIdentity)* are assigned to F4.0 Things (*relatesToPerson)* – in the sense of ownership (*ownedBy*), administration (*administratedBy)*, use (*usedBy*) by a person, or representation of a person (*represents*).

The latter is a special case of F4.0 Things that are F4.0 Components of a person and its Digital Twin. ***For persons,*** there are a total of three entries:

- *PersonIdentity* in S³I IdentityProvider – the identity of the person itself.

[7] https://www.rabbitmq.com/
[8] Passwords are only stored as hashes in Keycloak

- *ThingIdentity* in S³I IdentityProvider – the identity of the Digital Twin of the associated F4.0 Component of this person.
- *Thing* in S³I Directory – the entry of this person's F4.0 Component in the directory

For all other F4.0 Things (forestry machines, calculation services, apps ...) there are always exactly two entries:

- *ThingIdentity* in S³I IdentityProvider – the identity of the Digital Twin of the associated F4.0 Component of this F4.0 Thing (forestry engines, computation services, apps ...).
- *Thing* in S³I Directory – the entry of the F4.0 Component of this F4.0 Thing (forestry machines, calculation services, apps ...) in S³I Directory.

Additionally, for each F4.0 Thing there are entries in S³I IdentityProvider regarding ownership and administration as mentioned above.

2.2 S³I Directory

The right part of Figure 3 shows the ***conceptual data model of S³I Directory***. It starts at *root* that includes an API version. All things managed by S³I Directory (*Thing* via *things*) are accessible via this root node.

F4.0 Things (*Thing*) have a *type*, i.e. F4.0 Component (*component)*, F4.0 Service (*service*) or F4.0 HMI (*hmi*), and possess a mandatory *identifier (*identical to the *identifier of* the associated *ThingIdentity* in S³I IdentityProvider) as well as a *name*. In addition, they have a public key (*publicKey*) to allow for asymmetrically encrypted messages (see Section 4). Optionally, a *location* can be stored in S³I Directory and kept up to date in order to be able to make spatial requests for F4.0 Things (primarily, for physical assets). In addition, information about the data model used by the F4.0 Thing can be stored. This informs a communication partner, e.g., when directly accessing a DT via REST, about the expected data structures, e.g., in ForestML 4.0. Each F4.0 Thing can be related (via *childThings*) to its possible parts, if these are managed as independent F4.0 Things in S³I Directory, e.g., a timber truck and a crane mounted on it or an organization and its members. In addition, a default F4.0 HMI (*defaultHMI*) can be assigned to each F4.0 Thing, via which the F4.0 Thing (especially F4.0 Components) provides a user interface or a person can be reached by default. This can be used, for example, to define the on-board computer user interface (F4.0 HMI) of a harvester (F4.0 Component) or the default app (F4.0 HMI) of a forest manager (F4.0 Component).

Each F4.0 Thing can be assigned *endpoints* via *allEndpoints*, via which it can be reached and communicate with the respective selected transport protocol (OPC UA, REST, MQTT, S³I-B ...). Thus, S³I does not restrict the choice of protocols. Endpoints allow access to the complete structure or individual properties and services of the F4.0 Thing. Parts of these endpoints can additionally be defined as *default* endpoints, to which messages should be sent in case of doubt, in order to resolve ambiguities if necessary. Several default endpoints should have different protocols.

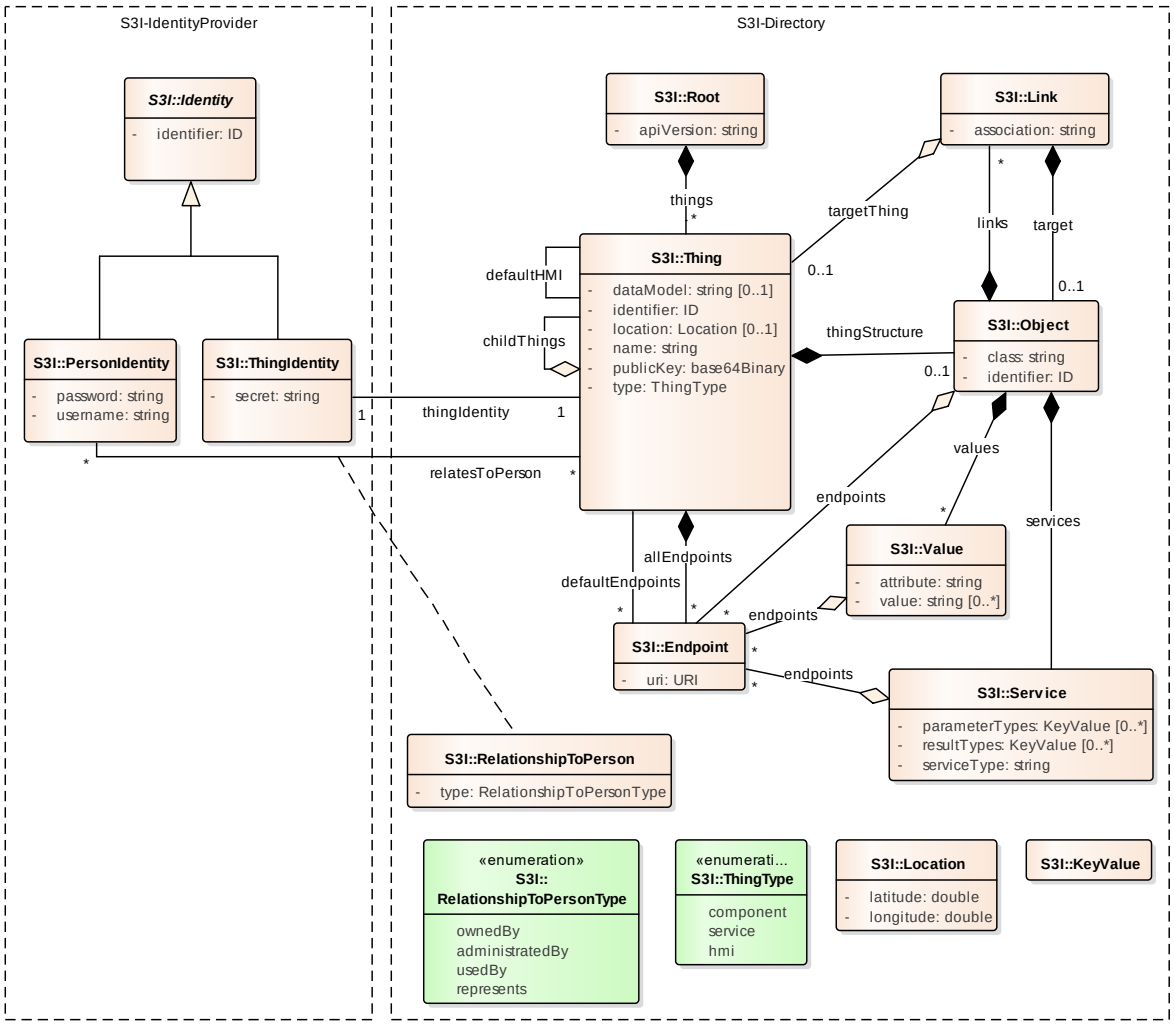

Figure 3: The conceptual data model of S³I IdentityProvider (left) and S³I Directory (right).

To specify this in more detail, the structure of an F4.0 Thing (or the relevant parts of it) can already be mapped in the S³I Directory (via *thingStructure*). For this purpose, a generic object-oriented data model consisting of objects (*Object*), values (*Value)*, provided services (*Service*) as well as object structures via links (*Link*) is provided. The genericity of this data model allows to represent arbitrary data models of Things. The respective instantiated elements of the concrete target data model are specified via the properties *Object::class*, *Value::attribute* as well as *Link::association.* The endpoints of the F4.0 Thing can then be assigned to the objects, values and services in detail in order to express that a specific endpoint, e.g., provides access to a sub-object or a sub-object tree, provides a very specific value of an asset (e.g., speed of the motor of a harvester) or provides a very specific service. A *Value* can also contain a cached *value in* addition to endpoints. A *service* can be used to describe standalone F4.0 Services as well as the functions provided by an F4.0 Component via its DT. The type of the service (*serviceType*) - also in the sense of a name - must be specified. In addition, the necessary input parameters (*parameterTypes*) as well as result types (*resultTypes)* can be defined as key-value pairs (*KeyValue).* The key describes the name of the parameter or partial result, the value the corresponding data type. Data types in both cases can be simple data types (int, boolean, string ...) as well as file formats (as MIME type).

2.2.1 JSON Mapping

The following sections show examples of S³I Directory entries and a ***mapping of the conceptual data model to JSON*** for the target system Eclipse Ditto as S³I Directory's basis. In general, objects are mapped to JSON objects (maps). Object properties become key-value pairs, with the attribute name

as the key and the attribute value as the value. The same is true for links (associations) with other objects. As a special case, sets of *KeyValue* instances (for *parameterTypes* and *resultTypes*) are each mapped to a JSON object and, thus, a map of the corresponding key-value pairs. Specific to Ditto, the *Thing::identifier* is mapped to the *thingId* because it has special semantics there. In the course of this, a *policyId* is added, which is set identically to the *thingId* and is used for the rights management of Ditto. All IDs of *ThingIdentities* are UUIDs with the namespace *s3i*, e.g., "s3i:ef39a0ae-1f4a-4393-9508-ad70a4d38a63". Similarly, Ditto requires that all specific properties of F4.0 Things be arranged under *attributes.* Overall, this results in the structure according to the following examples.

2.2.2 Example: F4.0 Component Harvester

The following describes the exemplary entry of an F4.0 Component of a harvester and its DT.


```
{
    "thingId": "s3i:ef39a0ae-1f4a-4393-9508-ad70a4d38a63",
    "policyId": "s3i:ef39a0ae-1f4a-4393-9508-ad70a4d38a63",
    "attributes": {
        "ownedBy": "606d8b38-4c3f-46bd-9482-86748e108f32",
        "name": "Harvester",
        "location": {
            "longitude": 10.117,
            "latitude": 20.136
        },
        "publicKey": "...",
        "type": "component",
        "dataModel": "fml40",
        "allEndpoints": [
            "s3ib://s3i:ef39a0ae-1f4a-4393-9508-ad70a4d38a63",
            "opcua://...",
            "mqtt://..."
        ],
        "thingStructure": {
            "class": "fml40.Harvester",
            "services": [{
                "serviceType": "fml40.Harvester.getProductionData",
                "endpoints": ["s3ib://s3i:ef39a0ae-1f4a-4393-9508-ad70a4d38a63"],
                "parameterTypes": {},
                "resultTypes": {"productionData": "application/stanford2010+hpr"}
            }],
            "links": [{
                "association": "fml40.Harvester.engine",
                "target": {
                    "class": "fml40.Engine",
                    "values": [{
                        "attribute": "fml40.Engine.rpm",
                        "endpoints": ["mqtt://..."]
                    }]
                }
            }]
        }
    }
}
```


Figure 4: Directory entry for an F4.0 Component harvester in JSON format.

In detail, the description of an F4.0 Component "Harvester" in Figure 4 comprises:

- A *thingId* (=identifier) "s3i:ef39..."
- A *policyId*
- All specific properties according to the conceptual data model of S³I Directory below *attributes*
- The specification of the machine's owner (*ownedBy*) as *identifier of* the associated *PersonIdentity* in S³I IdentityProvider "606d...".

- The display name (*name*) "Harvester".
- The spatial positioning (*location*) in the form of *longitude* / *latitude*
- A public key for secure communication (*publicKey*)
- The *type* of the F4.0 Thing, here *component*
- The data model (*dataModel*) ForestML 4.0 ("fml40") used by the F4.0 Thing, which describes the structure of the DT of the F4.0 Component
- The list of all endpoints (*allEndpoints*) of the F4.0 Component Harvester, in the example via S³I-B, OPC UA and MQTT
- A section of the hierarchical structure (*thingStructure*) of the DT of the F4.0 Component Harvester in the data model defined above
    - Starting with a root object (*Object*) of the *class* "fml40.Harvester".
    - and a service function (*services*) with name (*serviceType)* "fml40.Harvester.getProductionData", the specification of the associated endpoint (*endpoints*) "s3ib://s3i:ef39a0ae-1f4a-4393-9508-ad70a4d38a63", the specification of an empty parameter list (*parameterTypes*) as well as the return values (*resultTypes*) with a value "productionData" of type "application/stanford2010+hpr".
    - and a link (*links*) according to *association* "fml40.Harvester.engine" and a target object (*target*)
    - Target object of the *class* "fml40.Engine"
    - and a value (*values*) to attribute (*attribute)* "fml40.Engine.rpm" and indication of the associated endpoint (*endpoints)* "mqtt://...", via which this value is accessible

2.2.3 Example: F4.0 Service Calculate Stand Inventory Attribtues

In the following, the exemplary directory entry of an F4.0 Service for the calculation of stand inventory attributes is described.

```
{
    "thingId": "s3i:ab0717b0-e025-4344-8598-bccald3d5d47",
    "policyId": "s3i:ab0717b0-e025-4344-8598-bccald3d5d47",
    "attributes": {
        "ownedBy": "606d8b38-4c3f-46bd-9482-86748e108f32",
        "name": "Bestandesinventurattribute-Dienst",
        "type": "service",
        "publicKey": "...",
        "allEndpoints": [
            "s3ib://s3i:ab0717b0-e025-4344-8598-bccald3d5d47-1",
            "s3ib://s3i:ab0717b0-e025-4344-8598-bccald3d5d47-2"
        ],
        "thingStructure": {"services": [
            {
                "serviceType": "calculateStock",
                "endpoints": ["s3ib://s3i:ab0717b0-e025-4344-8598-bccald3d5d47-1"],
                "parameterTypes": {"surface": "application/zipped-shapefile"},
                "resultTypes": {"stock": "double"}
            },
            {
                "serviceType": "calculateStandAttributes",
                "endpoints": ["s3ib://s3i:ab0717b0-e025-4344-8598-bccald3d5d47-2"],
                "parameterTypes": {
                    "surface": "application/zipped-shapefile",
                    "referenceDate": "date"
                },
                "resultTypes": {"result": "application/x-forestgml"}
            }
        ]}
    }
}
```


Figure 5: Directory entry for an F4.0 Service for calculating stand inventory attributes

The service description in Figure 5 comprises:

- A *thingId* (=identifier) "s3i:ab07..."
- A *policyId*
- All specific properties according to the conceptual data model of S³I Directory below *attributes*
- The F4.0 Service owner's specification (*ownedBy*) as an *identifier of* the associated *PersonIdentity* in S³I IdentityProvider "606d..."
- The display name (*name*) "Bestandsinventurattribute-Dienst" ("Inventory attributes service").
- The *type* of the F4.0 Thing, here *service*
- A public key for secure communication (*publicKey*)
- The list of all endpoints (*allEndpoints*), in the example via S³I-B
- A section of the (here flat) structure (*thingStructure)* of the F4.0 Service
    - Starting with a type-less root object (*Object*)
    - two linked service functions (*services*)
    - one of type (*serviceType*) "calculateStock".
        - with *endpoint*(s) "s3ib://s3i:ab0717b0-e025-4344-8598-bcca1d3d5d47-1"
        - and specification of the parameters necessary for the call (*parameterTypes*)

- concretely, the parameter "surface" with expected data type "application/zipped-shapefile" (zipped ShapeFile), i.e., in the sense of a BASE64 encoded file
- and specification of the data types and names of the return values (*resultTypes*)
- concretely, the return value "stock" of the type "double

- another of type (*serviceType*) "calculateStandAttributes".
  - with *endpoint*(s) "s3ib://s3i:ab0717b0-e025-4344-8598-bcca1d3d5d47-2"
  - and specification of the parameters necessary for the call (*parameterTypes*)
  - concretely, the parameter "surface" with expected data type "application/zipped-shapefile" (zipped ShapeFile), i.e., in the sense of a BASE64 encoded file
  - and the parameter "referenceDate" with data type "date"
  - and specification of the data types and names of the return values (*resultTypes*)
  - specifically, the return value "standAttributes" of type "application/x-forest-gml" in terms of a BASE64 encoded ForestGML file

2.2.4 Example: F4.0 Component Forest Surveyor and his F4.0 HMI App

In the following, the exemplary entry of an F4.0 Component of a forest surveyor "Sachverstaendiger Schmitz" (i.e., a person) is described.

```
{
    "thingId": "s3i:2a2727eb-3be6-4c53-9300-6269a0969d43",
    "policyId": "s3i:2a2727eb-3be6-4c53-9300-6269a0969d43",
    "attributes": {
        "ownedBy": "800ee8e8-4873-4792-9294-c81948710938",
        "represents": "800ee8e8-4873-4792-9294-c81948710938",
        "name": "Sachverstaendiger Schmitz",
        "type": "component",
        "dataModel": "fml40",
        "defaultHMI": "s3i:6f58e045-fd30-496d-b519-a0b966f1ab01",
        "allEndpoints": [],
        "thingStructure": {
            "class": "fml40.Person",
            "values": {
                "attribute": "fml40.Person.occupation",
                "value": "consultant"
            },
            "links": [{
                "association": "fml40.Person.address",
                "target": {
                    "class": "fml40.Address",
                    "values": [
                        {
                            "attribute": "fml40.Address.name",
                            "value": "Schmitz"
                        },
                        {
                            "attribute": "fml40.Address.city",
                            "value": "Arnsberg"
                        }
                    ]
                }
            }]
        }
    }
}
```


Figure 6: Directory entry for an F4.0 Component of a forest surveyor in JSON format.

In detail, the description of the F4.0 Component in Figure 6 comprises:

- A *thingId* (=identifier) "s3i:2a27..."
- A *policyId*
- All specific properties according to the conceptual data model of the S³I Directory below *attributes*
- The specification of the owner of the F4.0 Component (*ownedBy*) as *identifier of* the associated *PersonIdentity* in the S³I IdentityProvider "800e..."
- The specification of the person (*PersonIdentity*) represented by the DT of this F4.0 Component (*represents*) as *identifier* of the associated *PersonIdentity* of the surveyor in S³I IdentityProvider "800e..." (here, identical to the owner)
- The display name (*name*) "Sachverstaendiger Schmitz".
- The *type* of the F4.0 Thing, here *component*
- The data model (*dataModel*) ForestML 4.0 ("fml40") used by the F4.0 Thing, which describes the structure of the DT of the F4.0 Component
- The indication that the F4.0 Thing with the *identifier* "s3i:6f58..." (i.e., the surveyor's app) is the default F4.0 HMI (*defaultHMI*) of this F4.0 Thing.
- The list of all endpoints (*allEndpoints*) of the F4.0 Component, in the example the list is empty because the surveyor can / wants to communicate exclusively via his app (F4.0 HMI)

- A section of the hierarchical structure (*thingStructure*) of the DT of the F4.0 Component in the data model defined above
    - Starting with a root object (*Object*) of the class (*class)* "fml40.Person".
    - a value (*values*) to attribute (*attribute)* "fml40.Person.occupation" and direct specification of the associated value "consultant" - without specification of an endpoint, the value is only available via S³I Directory
    - and a link (*links*) according to *association* "fml40.Person.address" and a target object (*target*)
    - Target object *of* the *class* fml40.Address
    - and a value (*values*) to attribute (*attribute)* "fml40.Address.name" and direct specification of the associated value "Schmitz" - without specification of an endpoint, the value is only available via S³I Directory
    - and a value (*values*) to attribute (*attribute)* "fml40.Address.city" and direct specification of the associated value "Arnsberg" - without specification of an endpoint, the value is only available via S³I Directory


```
{
    "thingId": "s3i:6f58e045-fd30-496d-b519-a0b966flab01",
    "policyId": "s3i:6f58e045-fd30-496d-b519-a0b966flab01",
    "attributes": {
        "ownedBy": "800ee8e8-4873-4792-9294-c81948710938",
        "administratedBy": "800ee8e8-4873-4792-9294-c81948710938",
        "usedBy": "800ee8e8-4873-4792-9294-c81948710938",
        "name": "App von Sachverstaendigem Schmitz",
        "type": "hmi",
        "publicKey": "...",
        "allEndpoints": ["s3ib://s3i:6f58e045-fd30-496d-b519-a0b966flab01"],
        "defaultEndpoints": ["s3ib://s3i:6f58e045-fd30-496d-b519-a0b966flab01"]
    }
}
```


Figure 7: Directory entry for the forest expert's F4.0 HMI app in JSON format.

The description of the F4.0 HMI described in Figure 7 includes:

- A *thingId* (=identifier) "s3i:6f58..."
- A *policyId*
- All specific properties according to the conceptual data model of S³I Directory below *attributes*
- The specification of the owner of the F4.0 HMI (*ownedBy*) as *identifier* of the corresponding *PersonIdentity* in S³I IdentityProvider "800e..." (here, the surveyor as a person)
- The specification of the administrators of the F4.0 HMI (*administratedBy*) as *identifier* of the associated *PersonIdentity* in S³I IdentityProvider "800e..." (here, the surveyor as a person)
- The specification of the use of the F4.0 HMI (*usedBy*) as *identifier* of the associated *PersonIdentity* in S³I IdentityProvider "800e..." (here, the surveyor as a person)
- The display name (*name*) "App von Sachverstaendigem Schmitz".
- The *type* of the F4.0 Thing, here *hmi*
- A public key for secure communication (*publicKey*)

- The list of all endpoints (*allEndpoints*) of the F4.0 HMI, in the example an S³I-B endpoint
- The list of default endpoints (*defaultEndpoints*) of the F4.0 HMI, in this example an S³I-B endpoint

2.3 S³I-B-Protokoll

Figure 8 shows the conceptual data model for the structure of a *message* for the S³I-B protocol of S³I Broker. A message comprises information about the sending F4.0 Thing (*sender*) in the form of an identifier from S³I Directory (*identifier* of a Thing instance, see Figure 3) and an identifier for the message (*identifier*). One or more *receivers* can be assigned to each message, also in the form of identifiers from S³I Directory.

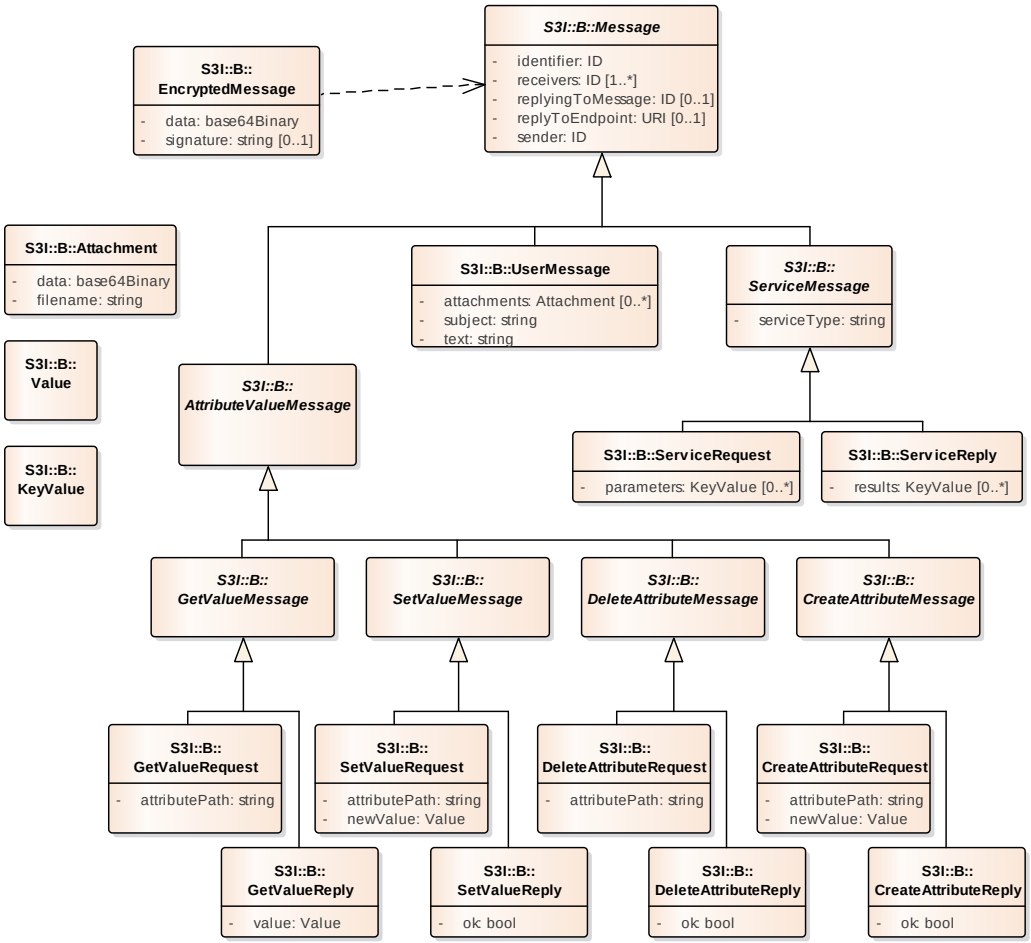

Figure 8: The conceptual data model for messages of the S³I-B protocol used for S³I Broker

An additional building block of each message can be the information whether it represents the reply to a previous message (*replyingToMessage* with its *identifier*) or to which endpoint possible replies should be sent back (*replyToEndpoint*).

Messages can be encrypted and signed in their entirety (*EncryptedMessage*, for details see Section 4).

Furthermore, there are three basic types of messages:

- Human readable message between users (*UserMessage*)
- Machine-readable message for accessing service functions of an F4.0 Service or an F4.0 Component (*ServiceMessage*)

  - to call (*ServiceRequest*) and
  - for the corresponding response (*ServiceReply*)
- Machine-readable message for accessing data (*AttributeValueMessage*) of the F4.0 Thing
  - For reading attribute values (*GetValueMessage*) with request (*GetValueRequest*) and response (*GetValueReply*).
  - For writing attribute values (*SetValueMessage*) with request (*SetValueRequest*) and response (*SetValueReply*).
  - To delete attributes (*DeleteAttributeMessage*) with request (*DeleteAttributeRequest*) and response (*DeleteAttributeReply*).
  - To create attributes (*CreateAttributeMessage*) with request (*CreateAttributeRequest*) and response (*CreateAttributeReply*).

A *UserMessage* includes the usual fields for e-mail-type communication for *subject*, *text* content, and *attachments*. An *attachment* combines file name (*filename*) with associated BASE64 encoded data (*data*).

The specifications for a *ServiceMessage* correspond to the specifications in Section 2.2 regarding the specification of F4.0 Service interfaces in S³I Directory. In general, the *serviceType* describes exactly which service function is to be called at the recipient. A call (*ServiceRequest*) then contains the necessary call *parameters* in the form of a list of key-value pairs with the parameter name as key and parameter value as value. Similarly, a service response (*ServiceReply*) contains the *results* as a list of key-value pairs.

An *AttributeValueMessage* can be used to access properties of an F4.0 Thing (usually, an F4.0 Component). This access to the properties of the F4.0 Thing (e.g., to the properties of a forestry machine by a Digital Twin provided on the forestry machine itself ("Edge")) itself is not to be confused with an access to the entry of the F4.0 Thing in S³I Directory. This requires different parameters in each case. In all variants, the *attributePath* specifies the path to the considered attribute of the F4.0 Thing. The *value* or *newValue* describes the read or new value to be written to this attribute. On modifying access, a flag (*ok*) returns the success status of the action as a response.

2.3.1 JSON Mapping

Examples of S³I-B messages as well as the ***mapping of the conceptual data model to JSON are*** shown in the following sections for a message between persons (2.3.2), a call to an F4.0 Service (2.3.3) and a response from an F4.0 Service (2.3.4) as well as the query of an attribute value of an F4.0 Component (2.3.5) and the corresponding response (2.3.6). Contents are exemplary or exemplary describing the actual content (in angle brackets).

In general, the mapping of the conceptual data model to JSON is the same as for S³I Directory. As with S³I Directory, *KeyValue* lists are mapped directly to a JSON object with key-value pairs in it. The type of a message, i.e. *UserMessage*, *ServiceRequest* or *ServiceReply*, *GetValueRequest* or *GetValueReply* etc., is mapped in JSON format via the *messageType* property.

2.3.2 Example: Message between Persons

Figure 9 shows a user message from a forest owner to his consultant – more precisely, between their respective F4.0 HMI (e.g., their apps). In detail, the message consists of:

- The *thingId/identifier of* the sending F4.0 Thing, here, the F4.0 HMI (app) of the forest owner.
- The *identifier* of the message itself

- The list of *receivers* (*receivers*) with a recipient (the consultant) in the form of the *thingId/identifier of* its F4.0 HMI (App) (*receiver*).
- The type of the message (*messageType*) – in this case a user message (*userMessage*).
- The specification of the endpoint to which the sender (forest owner) expects the message (*replyToEndpoint*), in this case the endpoint of his app
- A list of file *attachments* with a file "foto.jpg" (*filename*) and its BASE64 encoded content (*data*)
- The subject (*subject*)
- The text (*text*)


```
{
    "sender": "s3i:2aafd97c-ff05-42b6-8e4d-e492330ec959",
    "identifier": "s3i:1385e09e-3e93-4c2f-92d3-345698c40407",
    "receivers": ["s3i:6f58e045-fd30-496d-b519-a0b966flab01"],
    "messageType": "userMessage",
    "replyToEndpoint": "s3ib://s3i:2aafd97c-ff05-42b6-8e4d-e492330ec959",
    "attachments": [{
        "filename": "foto.jpg",
        "data": "..."
    }],
    "subject": "Ein Betreff",
    "text": "Hallo Herr Schmitz, ... Mit freundlichen Grüßen, Maier"
}
```


Figure 9: Exemplary S³I-B message between persons (JSON format)

2.3.3 Example: Service Call

Figure 10 shows an exemplary message for calling an F4.0 Service for stock calculation by an F4.0 HMI, in the example of an consultant's app. In detail, the message consists of:

- The *thingId/identifier of* the sending F4.0 Thing, here, the F4.0 HMI (app) of the consultant
- The *identifier* of the message itself
- The list of *receivers* with a receiver (the F4.0 Service) in the form of its *thingId/identifier* (*receiver*).
- The type of the message (*messageType*) – here a service call (*serviceRequest*).
- The specification of the endpoint to which the sender (consultant) expects the message (*replyToEndpoint*), in this case the endpoint of his app
- The specification of exactly which *service* function (*serviceType*) should be called, here "calculateStock".
- The specification of the necessary call parameters (*parameters*), here the parameter "surface" with a BASE64 encoded zipped ShapeFile.

```
{
    "sender": "s3i:6f58e045-fd30-496d-b519-a0b966f1ab01",
    "identifier": "s3i:08938da3-a2aa-47be-810c-9e6ee97e7fdb",
    "receivers": ["s3i:ab0717b0-e025-4344-8598-bccald3d5d47"],
    "messageType": "serviceRequest",
    "replyToEndpoint": "s3ib://s3i:6f58e045-fd30-496d-b519-a0b966f1ab01",
    "serviceType": "calculateStock",
    "parameters": {"surface": "..."}
}
```


Figure 10: Exemplary S³I-B message for calling an F4.0 Service (JSON format)

2.3.4 Example: Service Result

Figure 11 shows the response of the F4.0 Service for stock calculation to the consultant (or his app) from the previous section. In detail, the message consists of:

- The *thingId/identifier of* the sending F4.0 Thing, here the F4.0 Service
- The *identifier* of the message itself
- The list of *receivers* (*receivers*) with a receiver – the F4.0 HMI (app) of the expert – in the form of their *thingId/identifier* (*receiver*).
- The type of the message (*messageType*) – here, a service reply (*serviceReply*).
- The indication, which *service* function (*serviceType*) exactly was called, here, "calculateStock".
- The information on which message was replied to in the form of its *identifier*.
- The *results*, here the result "stock" with value "123.4" according to specification in S³I Directory


```
{
    "sender": "s3i:ab0717b0-e025-4344-8598-bccald3d5d47",
    "identifier": "s3i:f13ef9dd-68d8-43c9-b920-99c943c42089",
    "receivers": ["s3i:6f58e045-fd30-496d-b519-a0b966f1ab01"],
    "messageType": "serviceReply",
    "serviceType": "calculateStock",
    "replyingToMessage": "s3i:08938da3-a2aa-47be-810c-9e6ee97e7fdb",
    "results": {"stock": "123.4"}
}
```


Figure 11: Exemplary S³I-B message with the result of an F4.0 Service (JSON format)

2.3.5 Example: Attribute Value Query to a Digital Twin

Figure 12 shows an example for the request of an attribute value. The DT of a sawmill asks the DT of a forwarder for its properties ("features"). In detail, the message consists of:

- The *thingId/identifier* of the sending F4.0 Thing, here, the Digital Twin of the sawmill
- The *identifier* of the message itself
- The list of *receivers* with one receiver – here the DT of the forwarder – in the form of its *thingId/identifier* (*receiver*).
- The type of the message (*messageType*) – here the request of an attribute value (*getValueRequest*).
- The specification of the endpoint to which the sender expects the message (*replyToEndpoint*), here the S³I-B endpoint of the DT of the sawmill.
- The path to the relevant attribute, here, all properties at once ("features")

```
{
    "sender": "s3i:2073c475-fee5-463d-bce1-f702bb06f899",
    "identifier": "s3i:f5ea891c-530b-11ea-9a4e-74d02bc5ce32",
    "receivers": ["s3i:c81c7f54-46f4-40d6-a6e6-4bd0e8a7f08d"],
    "messageType": "getValueRequest",
    "replyToEndpoint": "s3ib://s3i:2073c475-fee5-463d-bce1-f702bb06f899",
    "attributePath": "features"
}
```


Figure 12: Exemplary S³I-B message for retrieving an attribute value of a forwarder (JSON format).

2.3.6 Example: Attribute Value Result from a Digital Twin

Figure 13 shows an example of the response of the forwarder's Digital Twin to the previous request for an attribute value from the sawmill. In detail, the message consists of:

- The *thingId/identifier of* the sending F4.0 Thing, here, the forwarder
- The *identifier* of the message itself
- The list of *receivers* with one receiver – here, the sawmill – in the form of their *thingId/identifier* (*receiver*).
- The type of the message (*messageType*) – here, the return of an attribute value (*getValueReply*).
- The indication of which message was replied to (*replyingToMessage*).
- The *value* of the queried attribute, in this case, all features of the forwarder, specifically the scale associated with the forwarder (abbreviated).


```
{
    "sender": "s3i:c81c7f54-46f4-40d6-a6e6-4bd0e8a7f08d",
    "identifier": "s3i:b42e395d-b519-4f50-8bcf-1cd32d47c7e9",
    "receivers": ["s3i:2073c475-fee5-463d-bce1-f702bb06f899"],
    "messageType": "getValueReply",
    "replyingToMessage": "s3i:f5ea891c-530b-11ea-9a4e-74d02bc5ce32",
    "value": [{
        "class": "ml40::Composite",
        "targets": [{
            "class": "ml40::Thing",
            "name": "Waage am Forwarder",
            "identifier": "...",
            "roles": [{"class": "ml40::Scale"}],
            "features": [{
                "class": "ml40::Weight",
                "currentWeight": "300"
            }]
        }]
    }]
}
```


Figure 13: Exemplary S³I-B message of the DT Forwarder with the result for retrieving an attribute value (JSON format).

2.4 S³I Repository

While S³I Directory provides a directory of all F4.0 Things including the (optional) description of their structure and their access points, S³I Repository, which is also based on Eclipse Ditto, can hold the data of the F4.0 Things themselves in a JSON-based manner. Analogous to S³I Directory, the basic Ditto system initially requires only a basic structure consisting of *thingId* and *policyId.* Properties are generally expected under the *attributes* (for more static) or *features* (for more variable) keys. Figure 14 shows this basic structure.

```
{
    "thingId": "s3i:7f654f13-11ac-413d-841c-10a8e431fc35",
    "policyId": "s3i:7f654f13-11ac-413d-841c-10a8e431fc35",
    "attributes": {},
    "features": {}
}
```


Figure 14: Basic structure of an entry in S³I Repository

For modeling F4.0 Things, KWH4.0 is developing the ***ForestML 4.0*** data model, which is described in more detail in a another KWH4.0 position paper. As in S³I Directory, the hierarchical description of an F4.0 Thing built according to the ForestML 4.0 data model is kept entirely below the *attributes* key to allow for a self-contained mapping. However, the Ditto base system provides optimized data storage and access to frequently changing properties below the *features* key. Therefore, KWH4.0 recommends an optional swapping of such values into the *features* area, which is then marked at the origin via a special keyword "*ditto-feature:idX*".

It is important to note that the same conceptual data model – for F4.0, KWH4.0 recommends ForestML 4.0 – maps slightly ***differently to*** the JSON structures in S³I Directory and S³I Repository, respectively:

- In ***S³I Directory,*** below the key *thingStructure,* a rather ***indirect mapping is*** done via the metamodel defined in Figure 3 consisting of *S3I::Object*, *S3I::Link*, S3I::*Value* and *S3I::Service*. By this approach, any data model (not only ForestML 4.0) can be mapped in S³I Directory. In addition, this approach also allows the necessary metadata of S³I Directory, especially, in the form of endpoints (*S3I::Endpoint*), to be linked to the corresponding structures. For example, an attribute value can be assigned an endpoint address via which the value can be retrieved. Likewise, the signature of a service interface can be defined in this way.

- In ***S³I Repository*** the (ForestML 4.0) data model is ***mapped*** rather ***directly*** to JSON. This means, attributes and associations in the data model directly become keys in the JSON document. Service functions, on the other hand, are not mapped at all because they do not describe any data and are already mapped via the data model description of the associated classes. Objects themselves, on the other hand, are specified identically as JSON objects with specification of the *class.*

2.4.1 Example: Wheel Loader

Figure 15 shows a section of a wheel loader modeled in ForestML 4.0 and stored in S³I Repository.

- It is identified by its *thingId* s3i:7f654f13-11ac-413d-841c-10a8e431fc35. This is the same ID as used in S³I Directory and S³I IdentityProvider for this F4.0 Thing.

- The access rights are specified via the policy with *policyId* s3i:7f654f13-11ac-413d-841c-10a8e431fc35.

- The complete hierarchical ForestML 4.0-compliant description of the F4.0 Thing is below the *attributes* key (block "1" in the figure). Note the difference between the *features* key defined by the Ditto base system and the *features* key of the same name defined by ForestML 4.0 (within block "1" in the figure), which have different meanings. For more details, see the KWH4.0 position paper on ForestML 4.0 mentioned above.

- For optimized data management and access by the Ditto base system, individual properties can be outsourced to Ditto features (block "2" in the figure). In the example, this is the property *currentLevel* of the tank of the wheel loader (block "1" in the figure). The value of this property is swapped out via the keyword "*ditto-feature:id3*" into a Ditto feature with the identifier "id3" (block "2" in the figure). Note: The other two swapped out features ("id1" and "id2") are referenced by the hidden part of the example ("[9 lines]").

Figure 15: Wheel loader modeled with ForestML 4.0 in S³I Repository (1: ForestML 4.0-specific part; 2: outsourced Ditto features for optimized data management).

3 Authorization in S³I Directory

Details on authorization in S³I Directory will follow in the next version of this position paper.

4 Encryption and Signing in the S³I-B Protocol

To ***encrypt*** and sign messages, the S³I-B protocol uses ***Pretty Good Privacy (PGP)*** (Figure 16). The URI scheme "s3ibs://" is used to identify S³I-B endpoints that support encrypted (and also unencrypted) messages. Endpoints that support only unencrypted messages shall use the URI scheme "s3ib://".

PGP is an encryption standard that provides cryptographic privacy and authentication for data communications. The Network Working Group describes the ***Open PGP Message Format*** in ***RFC 4880***. The format uses a combination of symmetric and asymmetric keys.

In symmetric key cryptography, a message is encrypted and decrypted with the same cryptographic keys. The two keys may have different formats, but they are based on the same "secret". This secret

must be shared between the parties involved, which requires a secure channel for exchanging the secret. Asymmetric cryptography avoids the shared secret. Each user provides a key pair consisting of a public key and a private key. Data encrypted with one of these keys can only be decrypted with the corresponding counterpart. Assuming that the private key is always kept private, encrypting data with the public key results in only the owner of the private key to be able to decrypt that data. In this way, using a person's public key ensures that only that person can decrypt the message, as long as the private key's affiliation with the expected recipient is preserved. In this way, the confidentiality of a PGP message is guaranteed.

The private key can not only be used for decryption, but also for ***signing*** messages. Again, only the expected owner of the private key uses the private key, so when data is signed with this key, you can be sure that this owner signed the message. The signature of a message is a hash that is encrypted with the private key. It is appended to the actual message. When the recipient decrypts the signature with the sender's respective public key, it should match the hash of the message that the sender himself has calculated. If the message is modified, the verification would fail because the signature is not based on the modified message, but on the original one, and is compared with the hash of the modified message. The use of such signatures ensures the authenticity ("Is the message really coming from the specified sender?") and integrity of the message.

In comparison between symmetric and asymmetric key cryptography, symmetric encryption is lighter, which means that it requires less computational resources, making it faster compared to asymmetric encryption. Nevertheless, it does not provide message authentication through signatures and is problematic in terms of key distribution. PGP combines the advantages of asymmetric and symmetric encryption by encrypting the data with a symmetric key, encrypting this key using a public-key cryptography approach, and sending it along with the data. In this way, the relatively resource- and time-consuming public-key encryption is applied only to the symmetric key, and the actual data encryption is faster and more resource-efficient with a symmetric key. The process is illustrated in Figure 16. If there are multiple recipients, the symmetric key must be encrypted for each recipient with their respective public key. The message can also be signed before encryption.

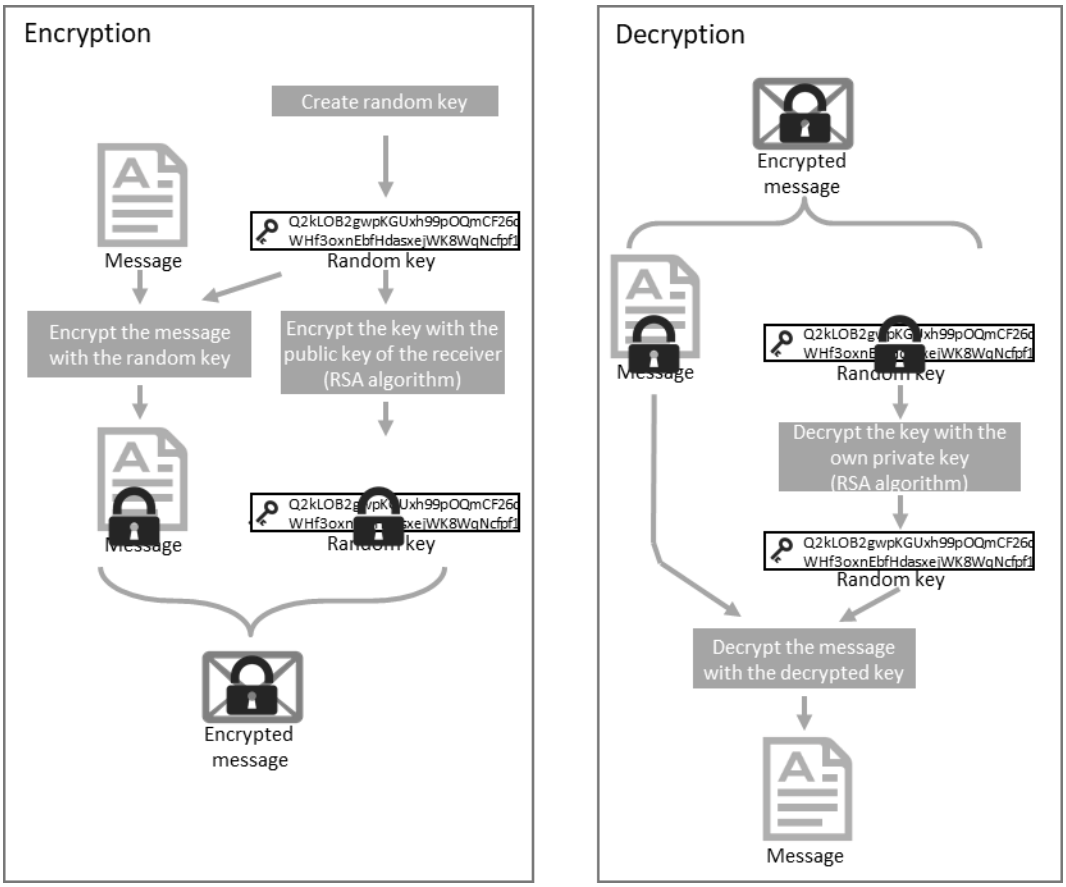

Figure 16: Encryption and decryption of S3I-B messages using the PGP method.[9]

The RFC standard does not prescribe a concrete encryption algorithm, but specifies which algorithms must, should and can be implemented by a PGP application. In S³I, the RSA (Rivest-Shamir-Adleman) algorithm with a 2048 bit long key is to be used for public key encryption and an AES (Advanced Encryption Standard) with a 512 bit long key for symmetric encryption.

5 Authentication with S³I IdentityProvider

In the context of S³I, the ***OpenID Connect (OIDC)*** standard is used for authentication. Specifically, two "login flows" are used.

5.1 Person und F4.0 HMI

If a ***person is*** to authenticate ***via an F4.0 HMI***, the "Authorization Code Flow" is used. In simple terms, the person logs on to the F4.0 HMI with their username and password (according to their *PersonIdentity*). The F4.0 HMI then authenticates itself to S³I IdentityProvider using this username and password (the *PersonIdentity)* and its own secret (its *ThingIdentity*) and receives a JWT in return. The JWT can then be used to authenticate the combination of person and F4.0 HMI to third parties.

5.2 F4.0 Thing without a Person

If, on the other hand, an ***F4.0 Thing*** is to operate ***independently*** without the direct involvement of a person, the "Client Credentials Flow" is used. Here, the F4.0 Thing only authenticates itself with its secret (its *ThingIdentity*) to S³I IdentityProvider and receives a JWT in return. The JWT can then be used to authenticate the F4.0 Thing to third parties.

[9] Figure based on *„How PGP encryption works“* (https://en.wikipedia.org/wiki/Pretty_Good_Privacy#/media/File:PGP_diagram.svg) of xaedes&jfreax&Acdx. License: CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0).

6 Use Cases

Three exemplary use cases for the interaction of S³I services are described below. These can also be reproduced with the enclosed code examples.

6.1 Message Exchange between Persons

A forest owner sends a confidential message to his consultant.

- The forest owner enters his login data, i.e., username and password of his personal identity (*PersonIdentity*), into an F4.0 HMI – as an example, an app
    - The app authenticates itself to S³I IdentityProvider and receives a JWT according to Section 5.1
- The forest owner opens a dialog in his app to write secure S³I-B messages and searches for the last name of his consultant
    - The app searches S³I Directory for the last name entered and finds the associated F4.0 Component of the consultant. To do this, the app authenticates itself with S³I Directory using the JWT.
    - Since this F4.0 Component of the consultant does not have its own endpoint, the app asks S³I Directory for the default F4.0 HMI (*defaultHMI*) of this F4.0 Component. In the example, this entry points to an app of the consultant.
    - From this F4.0 HMI, it selects the default S³I-B endpoint (from *defaultEndpoints*).
- The forest owner writes a message with subject and text and attaches a photo as a file attachment
- The forest owner presses send
    - The app forms a JSON-based S³I-B user message according to Section 2.3.2
        - With *sender* as *identifier* of the app of the forest owner (a F4.0 HMI) and *receiver* as *identifier* of the app of the consultant (also an F4.0 HMI)
        - Under *replyToEndpoint* it enters its own S³I-B endpoint so that replies are sent back directly to the app (if it should not be the default F4.0 HMI or its default endpoint)
    - The app signs and encrypts the message according to Section 4
    - The app authenticates itself with the JWT at S³I Broker
    - The app passes the encrypted message via the AMQP protocol to the queue of S³I Broker stored in the URI
- The consultant receives a notification from the F4.0-compliant app on his smartphone about a new message
    - The app authenticates itself to S³I IdentityProvider and receives a JWT according to Section 5.1
    - The app authenticates itself to S³I Broker using the JWT
    - For this purpose, S³I Broker notifies the F4.0 HMI of the consultant, in the example also an app, about the new message in the queue

- The consultant retrieves the new message and reads it
    - The app retrieves the new message from its queue
    - The app checks the signing of the message content and decrypts it according to Section 4

6.2 Calling a Stock Calculation Services

A consultant wants to have the stock calculated for an area via remote sensing methods from an F4.0 Service.

- The surveyor uses a GIS-enabled app for this purpose as F4.0 HMI
- The consultant enters his login data, i.e., username and password of his personal identity (*PersonIdentity*), into the app
    - The app authenticates itself to S³I IdentityProvider and receives a JWT according to Section 5.1
- The consultant outlines the desired area and calls a service call function for it in his app
    - The app authenticates itself with its JWT at S³I Directory
    - The app searches S³I Directory for F4.0 Services with *"serviceType": "calculateStock"*.
    - The app finds a matching F4.0 Service
    - The app retrieves the parameter description (*parameterTypes*) of the corresponding *service* entry and recognizes that the (only) parameter *surface* should be transmitted as a zipped ShapeFile (*"surface": "application/zipped-shapefile"*) and provides the ring accordingly
    - The app retrieves the URI of the default S³I-B endpoint of this *service* entry (from *defaultEndpoints*).
    - The app forms a JSON-based S³I-B message with a service call according to Section 2.3.3
        - With *sender* as *identifier* of the F4.0 HMI (app) and *receiver* as *identifier* of the F4.0 Service
        - With the specification of their S³I-B endpoint for the reply (*replyToEndpoint*).
    - The app signs and encrypts the message according to Section 4
    - The app authenticates itself with its JWT to S³I Broker
    - The app passes the encrypted message via the AMQP protocol to the queue of S³I Broker stored in the URI
- In advance, once at its launch: The F4.0 Service for stock calculation ...
    - authenticates itself to S³I IdentityProvider and receives a JWT according to section 5.2
    - connects to S³I Broker,
    - authenticates with it using its JWT
- In operation when the message arrives from the app of the consultant: The F4.0 Service for stock calculation ...

- is informed by S³I broker about the new message in its queue
- retrieves the new message from its queue at S³I Broker
- checks the signing and decrypts the message according to Section 4
- parses the service call (*ServiceRequest*) and reads out the parameter data (*parameters*)
- executes its functionality (stock calculation)
- packages the *result* as a floating-point number (*"stock": 123.4*) into a JSON-based S³I-B message with a service result (*ServiceReply*) according to Section 2.3.4
- signs and encrypts the message according to Section 4
- reads the URI for the reply from the *replyToEndpoint* field of the incoming message
- passes the message to the queue of S³I Broker stored in this URI

- The consultant receives a notification from his app, retrieves it and gets the result displayed
  - The broker notifies the app of the consultant about the new message in the queue
  - The app retrieves the new message from its queue
  - The app checks the signing of the message content and decrypts it according to Section 4
  - The app identifies the message as a result (*ServiceReply*) of a previous service call and displays the result (*results*) of the ordered stock calculation

6.3 Querying a Property Value from a Forestry Machine

A sawmill (or the DT of the F4.0 Component sawmill) requires a StanForD 2010 data set from a harvester (or from the DT of the F4.0 Component harvester), which provides a corresponding service (as a functionality of the DT) for this purpose.

- The DT Sawmill ...
  - authenticates itself to S³I IdentityProvider and receives a JWT according to Section 5.2
  - authenticates itself with the JWT against S³I Directory
  - accesses the entry of the DT Harvester already known to it by *identifier* and searches in its structure (*thingStructure*) for a *service* entry with *"serviceType": "fml40.Harvester.getProductionData"* with empty parameter list (*parameterTypes*) and result type (*resultTypes*) *"productionData": "application/stanford2010+hpr"*
  - retrieves the URI of the S³I-B endpoint of this service (*Service::endpoints*)
  - forms a JSON-based S³I-B message with a service call (*ServiceRequest*) according to Section 2.3.3
    - With *sender* as *identifier* of the F4.0 Component sawmill and *receiver* as *identifier of* the F4.0 Component harvester
    - By specifying its S³I-B endpoint for the reply (*replyToEndpoint*).
  - signs and encrypts the message according to Section 4
  - authenticates with the JWT at S³I Broker

    - passes the encrypted message to the queue of the broker stored in the URI
- The DT Harvester ...
    - authenticates itself to S³I IdentityProvider and receives a JWT according to Section 5.2
    - connects to S³I Broker and authenticates itself with the JWT
    - is informed by S³I Broker about the new message in its queue
    - retrieves the new message from its queue at S³I Broker
    - checks the signing and decrypts the message according to Section 4
    - parses the service call (*ServiceRequest*) and reads out the parameter data (*parameters*)
    - Executes the selected functionality (*serviceType*) (compiles StanForD 2010 production data as HPR file).
    - packages the result (*results* with *"productionData": "<BASE64-encoded HPR file content>"*) into a JSON-based S³I-B message with a service result (*ServiceReply*) according to Section 2.3.4
    - signs and encrypts the message according to Section 4
    - reads the URI for the reply from the *replyToEndpoint* field of the incoming message
    - transfers the encrypted message to the queue of S³I Broker stored in the URI
- The DT Sawmill ...
    - is notified by S³I Broker
    - retrieves the new message from its queue
    - checks the signing of the message content and decrypts it according to Section 4
    - parses the message (*ServiceReply*) and assigns the *result* internally to the corresponding job (via *replyingToMessage*)