Smart Systems Service Infrastructure (S^3I)

Konzeption und Einsatz der Smart Systems Service Infrastructure (S^3I) zur dezentralen Vernetzung in Wald und Holz 4.0

Ein KWH4.0-Standpunkt

28.02.2020

Kompetenzzentrum Wald und Holz 4.0
c/o RIF Institut für Forschung und Transfer e.V. (Projektkoordination)
Joseph-von-Fraunhofer-Straße 20
D-44227 Dortmund
www.kwh40.de

Kontakt

Kompetenzzentrum Wald und Holz 4.0
c/o RIF Institut für Forschung und Transfer e.V. (Projektkoordination)
Joseph-von-Fraunhofer-Straße 20
D-44227 Dortmund
www.kwh40.de

Ansprechpartner: Dipl.-Ing. Frank Heinze
Tel. +49 (0) 231 9700-781
frank.heinze@rt.rif-ev.de
Verantwortlicher Autor: Dr. Martin Hoppen, MMI

Autoren

RIF Institut für Forschung und Transfer e.V. (Koordinator)
Geschäftsführer: Dipl.-Inf. Michael Saal
Joseph-von-Fraunhofer Str. 20, 44227 Dortmund

Werkzeugmaschinenlabor (WZL), RWTH Aachen
Institutsleiter: Prof. Dr.-Ing. Christian Brecher
Steinbachstraße 19, 52074 Aachen

Institut für Mensch-Maschine-Interaktion (MMI), RWTH Aachen
Institutsleiter: Prof. Dr.-Ing. Jürgen Roßmann
Ahornstraße 55, 52074 Aachen

Institut für Arbeitswissenschaft (IAW), RWTH Aachen
Institutsleiterin: Prof. Dr.-Ing. Verena Nitsch
Bergdriesch 27, 52062 Aachen

Wald und Holz NRW, Lehr- und Versuchsforstamt Arnsberger Wald
Forstliches Bildungszentrum für Waldarbeit und Forsttechnik
Leitung: FD Thilo Wagner
Alter Holzweg 93, 59755 Arnsberg

Förderhinweise

Dieses Vorhaben wird gefördert durch das Land Nordrhein-Westfalen unter Einsatz von Mitteln aus dem Europäischen Fonds für regionale Entwicklung (EFRE).

EFRE.NRW Investitionen in Wachstum und Beschäftigung
EUROPÄISCHE UNION Investition in unsere Zukunft Europäischer Fonds für regionale Entwicklung

Gemeinsam mit dem Vorhaben „iWald“, gefördert durch: Bundesministerium für Ernährung und Landwirtschaft aufgrund eines Beschlusses des Deutschen Bundestages (FKZ 22012718).

Bundesministerium für Ernährung und Landwirtschaft
FNR Fachagentur Nachwachsende Rohstoffe e.V.

Version	Datum	Abschnitte	Änderungen
1.0	19.12.2019	Alle	Erste offizielle Version
2.0	28.02.2020	1.4, 2.3+4, 4	S^3I-Config, S^3I-Repository, S^3I-B-Nachrichten

Smart Systems Service Infrastructure (S³I)

Grundlage der Kommunikation in Wald und Holz 4.0 ist die dezentrale Vernetzung in einem Internet der Dinge (Internet of Things, IoT). Diese ***WH4.0-Dinge*** umfassen ***WH4.0-Komponenten*** (Assets wie Maschinen, Geräte, Personen und ihre Digitalen Zwillinge), ***WH4.0-Dienste*** (Softwaredienste) und ***WH4.0-Mensch-Maschine-Schnittstellen*** (***WH4.0-MMS***) (Desktop-/Webapplikationen, Apps …), die situationsspezifisch zu ***WH4.0-Systemen*** miteinander vernetzt werden (siehe Abbildung 0-1), um ***Wertschöpfungsketten*** und damit Wertschöpfungsprozesse abzubilden. Diese WH4.0-Dinge können dabei durch verschiedene technologische Ansätze abgebildet werden. Digitale Zwillinge können ihre Laufzeitumgebung entweder direkt am bzw. auf dem betroffenen Asset haben (***Edge-Ansatz***) oder auf einem Server betrieben werden (***Cloud-Ansatz***); dazwischen gibt es Mischformen (***Fog-Ansätze***). Die Ansätze haben abhängig von Asset und Einsatzszenario verschiedene Vor- und Nachteile. Forstmaschinen wie Harvester profitieren im Einsatz vom Edge-Ansatz, weil hier Daten lokal verarbeitet und dann erst über Funkverbindungen übertragen werden. Assets, die keine Maschine darstellen (Wald, Polter, aber auch Geodaten) hingegen benötigen eine Laufzeitumgebung (Hard- und Software) in der Cloud. Dieselben Überlegungen lassen sich für Dienste anstellen (lokaler Lokalisierungs-Dienst im Edge-Ansatz gegenüber Waldwachstumssimulations-Dienst in der Cloud) oder Mensch-Maschine-Schnittstellen (lokale App auf dem Smartphone gegenüber Webapplikation in der Cloud). Weitere Details dazu liefert das KWH4.0 in Kürze in einem Standpunkt zur Architektur.

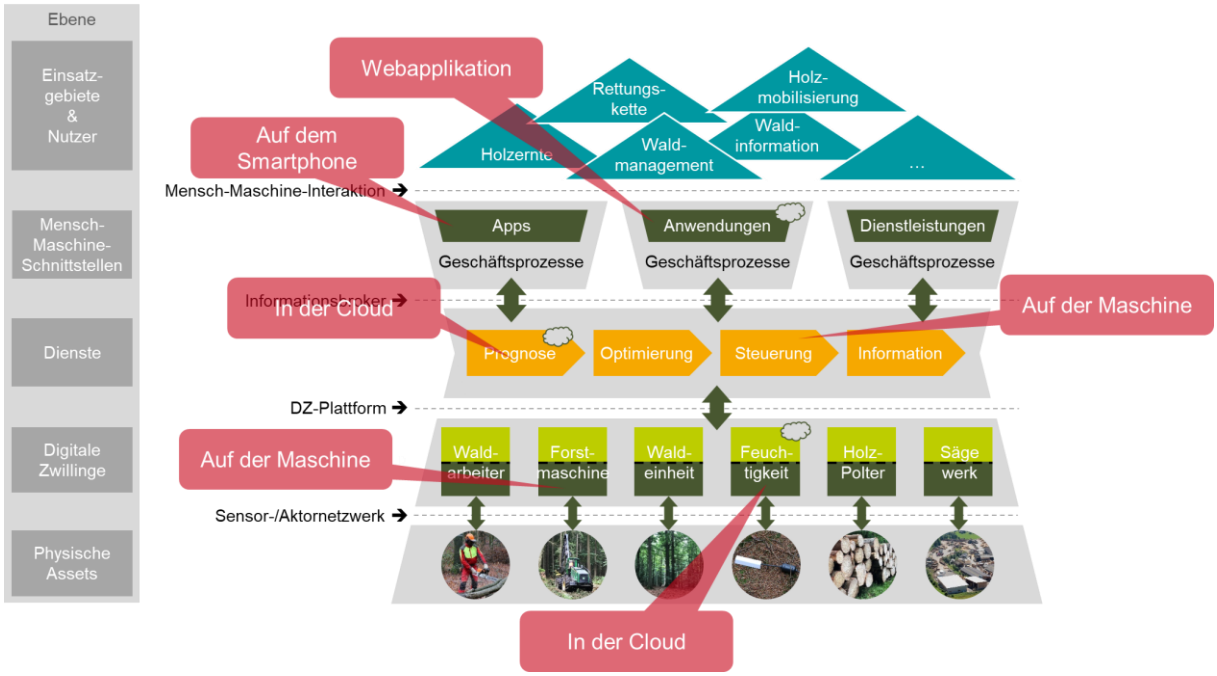

Abbildung 0-1: Vernetzung der „Dinge“ in Wald und Holz 4.0 und ihre Verortung auf der Maschine (Edge) oder in der Cloud[1]

Neben dieser technisch motivierten Heterogenität gibt es diese im Cluster Wald und Holz auch auf organisatorischer Ebene. Verschiedene Akteure möchten ihre WH4.0-Dinge typischerweise nicht auf einer zentralen Plattform betreiben, sondern ihre eigene Wahl treffen.

[1] Fotos (v. l.): A. Böhm, RIF; F. Heinze, RIF; S. Wein, WZL; A. Böhm; S. Wein; Michael Lorenzet / pixelio

Damit in diesem dezentralen IoT (bzw. Internet der WH4.0-Dinge) eine Vernetzung und damit Kommunikation möglich ist, benötigt man eine zentrale Infrastruktur bestehend aus wenigen zentralen Softwarediensten (diese sind selbst keine WH4.0-Dienste), über die sich WH4.0-Dinge möglichst nur einmalig authentifizieren, gegenseitig finden und miteinander kommunizieren können – unter Berücksichtigung der ihnen jeweils zugewiesenen Berechtigungen. Das KWH4.0 entwickelt für diese Zwecke die ***Smart Systems Service Infrastructure (S³I)*** und stellt sie allen interessierten Partnern bereit.

Im folgenden Abschnitt 1 werden die grundlegenden Konzepte von S³I erläutert. Abschnitt 2 geht dann im Detail auf die verwendeten Datenmodelle ein. Abschnitt 3 beschreibt die Autorisierung im Verzeichnis, Abschnitt 4 beschreibt das Verschlüsselungskonzept für Nachrichten, Abschnitt 5 die Authentifizierung und Abschnitt 6 zeigt Anwendungsfälle für die Verwendung von S³I. Dieses Standpunktpapier wird zudem ergänzt um verschiedene Programmierbeispiele, die Sie unter https://git.rwth-aachen.de/kwh40/s3i finden.

1 S³I-Konzept

Der grundlegende Aufbau von S³I ist in Abbildung 1-1 dargestellt.

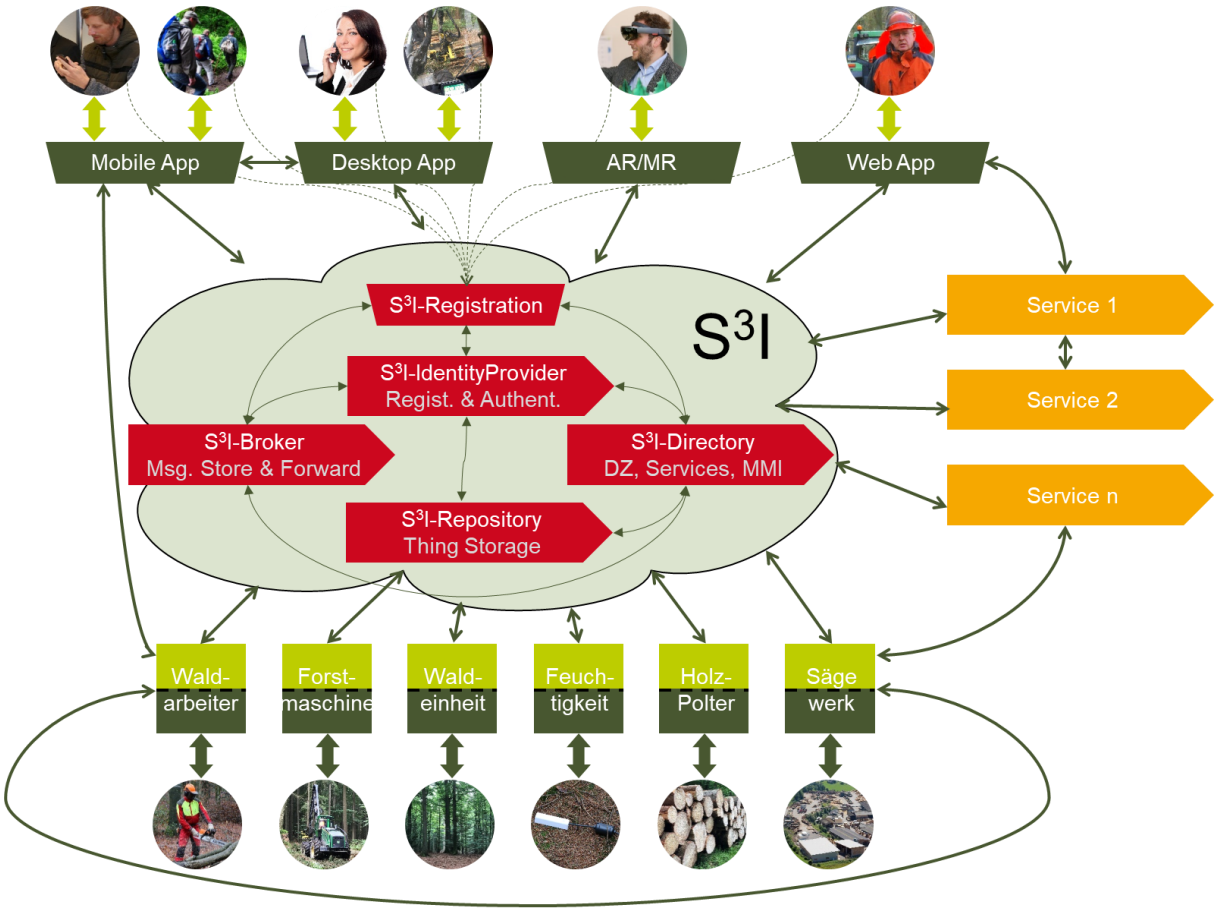

Abbildung 1-1: Grundkonzept der Smart Systems Service Infrastructure[2]

1.1 S³I-Directory

Das ***S³I-Directory*** ist das Kernelement von S³I und stellt einen Verzeichnisdienst dar, in dem Informationen über alle WH4.0-Dinge verwaltet werden. Hier lassen sie sich registrieren, mit Informationen

[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

versehen und verändern, über verschiedene Eigenschaften suchen und bei Bedarf wieder löschen. Die Authentisierung/Authentifizierung (als Antwort auf die Fragen „Wer bist Du?“ und „Bist Du wirklich der, der Du vorgibst zu sein?“) ist dabei Aufgabe des ***S³I-IdentityProviders*** (s.u.). Der Zugriff auf die Einträge des Directories wird über Policies rollenbasiert autorisiert (wodurch die Frage „Was darfst Du?“ berücksichtigt wird). Hier kann detailliert bis auf die Ebene einzelner Einträge festgelegt werden, wer auf welche Information lesend, schreibend oder löschend zugreifen darf (vgl. Abschnitt 3)

Die Informationen zu den WH4.0-Dingen im S³I-Directory umfassen allgemeine Angaben zu deren Identifikation, Typ, Rollen, wesentlichen Attributen, (häufig) räumlicher Verortung und zum verwendeten Datenmodell. Zentraler Bestandteil sind zudem Angaben zur Erreichbarkeit des jeweiligen WH4.0-Dings in Form sogenannter Endpoints über Protokolle wie OPC UA, MQTT, REST oder über das durch den ***S³I-Broker*** bereitgestellte, Nachrichten-basierte ***S³I-B-Protokoll*** (s.u.). Der genaue Aufbau des Verzeichnisses wird in Kapitel 2 dieses Standpunkts beschrieben. Die Authentifizierung am S³I-Directory erfolgt über JSON Web Token (JWT)[3], die über den S³I-IdentityProvider bereitgestellt werden.

Die Software-technische Umsetzung des S³I-Directories erfolgt über Eclipse Ditto[4], einer auch industriell eingesetzten Open-Source-Plattform für Digitale Zwillinge mit MongoDB-Backend und einer Policy-Infrastruktur zur Verwaltung der Autorisierungsdaten in S³I.

1.2 S³I-IdentityProvider

Der ***S³I-IdentityProvider*** stellt den zentralen Dienst für die Registrierung und Verwaltung von Identitäten sowohl von Personen selbst als auch von WH4.0-Dingen dar. Er ermöglicht Authentisierung („Ich bin es!“), bietet Authentifizierung („Ist er es wirklich?“) und erlaubt dabei ein Single-Sign-On (SSO). Dazu verwaltet er für jede Identität Benutzernamen, Passwort (für Personen) bzw. ein Geheimnis (für WH4.0-Dinge). Nach einmaliger Anmeldung am S³I-IdentityProvider mit Benutzername und Passwort bzw. mit Geheimnis stellt dieser zur Authentifizierung gegenüber anderen ein JSON Web Token aus. Dieses JWT kann sowohl für die Authentifizierung gegenüber den Diensten des S³I (Directory und Broker) als auch gegenüber beliebigen WH4.0-Dingen genutzt werden (sofern diese dies unterstützen können und möchten). Insgesamt wird dadurch ein SSO im gesamten WH4.0-IoT ermöglicht.

Die Umsetzung des S³I -IdentityProviders erfolgt über die auch industriell eingesetzte Open-Source-Software Keycloak[5].

1.3 S³I-Broker

Der ***S³I-Broker*** bietet allen WH4.0-Dingen einen in S³I integrierten Dienst für eine Nachrichten-basierte Kommunikation, vergleichbar mit einer formalisierten E-Mail. Er stellt einen optionalen Baustein von S³I dar, der neben anderen Transportprotokollen wie OPC UA, MQTT oder REST von einem WH4.0-Ding zur Vernetzung genutzt werden kann, aber nicht muss. Vorteil des integrierten Ansatzes ist, dass Dinge mit relativ geringem Aufwand mit anderen kommunizieren können. Durch die Nutzung formalisierter Nachrichten wird das ***S³I-B-Protokoll*** realisiert. Neben der Formalisierung der Struktur (vgl. Abschnitt 2.3) gehört auch ein integrierter Ansatz zur asymmetrischen Verschlüsselung dazu, der über im S³I-Directory hinterlegte öffentliche Schlüssel realisiert wird. Nachrichten können dabei in allen Richtungen zwischen allen WH4.0-Dingen wie z.B. zwischen Menschen (genauer: deren MMS), zwischen Mensch und Maschine (DZ oder Dienst) sowie untereinander zwischen Maschinen und zwischen Diensten genutzt werden. Der dabei realisierte Store&Forward-Ansatz löst das Problem der häufigen Nicht-Erreichbarkeit von WH4.0-Dingen im IoT von Wald und Holz 4.0. Andere klassische Industrie 4.0- oder

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

IoT-Protokolle sind dafür weniger geeignet, da sie verbindungsorientiert arbeiten und daher eine permanente Kommunikationsverbindung benötigten.

Technische Grundlage des S³I-Brokers ist das Advanced Message Queuing Protocol (AMQP)[6]. In Zusammenspiel mit einem AMQP-Broker können darüber asynchron zunächst beliebige Nachrichten ausgetauscht werden, die in sogenannten Queues für die adressierten WH4.0-Dinge gespeichert werden.

Die technische Umsetzung des S³I-Brokers basiert auf dem auch industriell eingesetzten Open-Source-AMQP-Broker RabbitMQ[7].

1.4 S³I-Registration

Während die zuvor genannten Dienste selbst nur maschinengeeignete Schnittstellen bieten, erlaubt die Webanwendung ***S³I-Registration*** Personen den Zugriff auf S³I, um darin

- ihre eigene Identität sowie die Identitäten der von ihnen verwalteten WH4.0-Dinge sowie
- diese WH4.0-Dinge selbst

zu verwalten (anlegen, verändern, löschen). Im Hintergrund greift die S³I-Registration dabei auf die einzelnen S³I-Dienste zu.

Die eigentliche technische Umsetzung der Webanwendung ist derzeit noch in der Konzeptionsphase.

Es steht bereits ein entsprechender Dienst ***S³I-Config*** zur Verfügung. Über eine REST API ermöglicht er die Registrierung und das Löschen von Personen sowie das Anlegen und Löschen von WH4.0-Dingen. ***S³I-Config*** stellt dabei sicher, dass diese Aktionen koordiniert in allen drei Teil-Diensten (IdentityProvider, Directory und Broker) des S³I durchgeführt werden.

1.5 S³I-Repository

Als Ergänzung bietet S³I das ***S³I-Repository*** als Cloud-Speicher für WH4.0-Dinge an. In diesem (ebenso wie dem S³I-Broker) optionalen Zusatzangebot können insbesondere Digitale Zwillinge gehostet werden.

Die technische Grundlage für das S³I-Repository ist dabei (wie beim S³I-Directory) Eclipse Ditto. Anders als das S³I-Directory werden hier aber nicht nur die „Telefonbucheinträge“ für WH4.0-Dinge verwaltet, sondern die WH4.0-Dinge selbst gespeichert. Dementsprechend handelt es sich um eine eigenständige Ditto-Instanz, neben der für das S³I-Directory.

2 S³I-Datenmodelle

In diesem Abschnitt werden die für das S³I-Directory, für den S³I-IdentityProvider sowie für das S³I-B-Protokoll verwendeten konzeptuellen Datenmodelle und ihre Abbildung auf JSON-Strukturen definiert.

2.1 S³I-IdentityProvider

Der linke Teil von Abbildung 2-1 zeigt das ***konzeptuelle Datenmodell des S³I-IdentityProviders***. Es umfasst die Verwaltung von Identitäten (*Identity*), die über einen Identifier (*identifier*) identifiziert werden können. Dabei wird unterschieden zwischen solchen Identitäten, die ein WH4.0-Ding beschreiben

[6] https://www.amqp.org/
[7] https://www.rabbitmq.com/

(*ThingIdentity*) und solchen, die eine Person, die S³I nutzt (*PersonIdentity*), beschreiben. Personen nutzen wie zuvor beschrieben Benutzername (*username*) und Passwort[8] (*password*) zur Anmeldung während WH4.0-Dinge ein Geheimnis (*secret*) nutzen.

Dabei bestehen Verknüpfungen mit dem Datenmodell des S³I-Directory. Jedem WH4.0-Ding im S³I-Directory ist zunächst genau eine *ThingIdentity* im S³I-IdentityProvider zugewiesen (über die Beziehung *thingIdentity*), über die es identifiziert wird. Zusätzlich werden Personen (*PersonIdentity*) WH4.0-Dingen (*Thing*) zugeordnet (*relatesToPerson*) – im Sinne des Besitzes (*ownedBy*), der Administration (*administratedBy*), bzw. der Nutzung (*usedBy*) durch eine Person oder der Repräsentation (*represents*) einer Person.

Letzteres ist ein Spezialfall für WH4.0-Dinge, die WH4.0-Komponenten aus einer Person und seinem Digitalen Zwilling darstellen. ***Für Personen*** gibt es insgesamt drei Einträge:

- *PersonIdentity* im S³I-IdentityProvider – die Identität der Person selbst
- *ThingIdentity* im S³I-IdentityProvider – die Identität des Digitalen Zwillings der zugehörigen WH4.0-Komponente dieser Person
- *Thing* im S³I-Directory – der Eintrag der WH4.0-Komponente dieser Person im Verzeichnis

Für alle sonstigen WH4.0-Dinge (Forstmaschinen, Berechnungsdienste, Apps …) gibt es immer genau zwei Einträge:

- *ThingIdentity* im S³I-IdentityProvider – die Identität des Digitalen Zwillings der zugehörigen WH4.0-Komponente dieses WH4.0-Dings (Forstmaschinen, Berechnungsdienste, Apps …)
- *Thing* im S³I-Directory – der Eintrag der WH4.0-Komponente dieses WH4.0-Dings (Forstmaschinen, Berechnungsdienste, Apps …) im S³I-Directory

Darüber hinaus gibt es für jedes WH4.0-Ding die oben genannten Einträge im S³I-IdentityProvider hinsichtlich Besitz und Administration.

2.2 S³I-Directory

Der rechte Teil von Abbildung 2-1 zeigt das ***konzeptuelle Datenmodell des S³I-Directories***. Der Einstieg erfolgt dabei über die Wurzel (*Root*) mit einer API-Version. Über diesen Wurzelknoten sind alle vom S³I-Directory verwalteten Dinge (*Thing* via *things*) erreichbar.

WH4.0-Dinge (*Thing*) haben einen *type* – also WH4.0-Komponente (*component*), WH4.0-Dienst (*service*) oder WH4.0-MMS (*hmi*)) – und besitzen einen obligatorischen *identifier* (identisch zum *identifier* der zugehörigen *ThingIdentity* im S³I-IdentityProvider) sowie einen Namen (*name*). Zudem verfügen sie über einen öffentlichen Schlüssel (*publicKey*), um dem WH4.0-Ding asymmetrisch verschlüsselte Nachrichten schicken zu können (vgl. Abschnitt 4). Optional kann im S³I-Directory eine Verortung (*location*) abgelegt und aktuell gehalten werden, um auch räumliche Anfragen nach WH4.0-Dingen (vornehmlich nach physischen Assets) stellen zu können. Zudem kann eine Information über das vom WH4.0-Ding genutzte Datenmodell hinterlegt werden. Dies informiert einen Kommunikationspartner z.B. beim Direktzugriff auf einen DZ per REST über die zu erwartenden Datenstrukturen, bspw. in ForestML 4.0[9]. Jedes WH4.0-Ding kann (über *childThings*) in Beziehung zu seinen etwaigen Teilen gesetzt werden, wenn diese als eigenständige WH4.0-Dinge im S³I-Directory verwaltet werden, bspw. ein Holz-LKW und ein daran montierter Kran oder eine Organisation und ihre Mitglieder. Zudem kann jedem WH4.0-Ding eine Standard-WH4.0-MMS (*defaultHMI*) zugeordnet werden, über die das WH4.0-Ding

[8] Die Passwörter werden in Keycloak natürlich nur als Hash gespeichert
[9] Vgl. Umsetzungsstrategie des KWH4.0

(insbesondere WH4.0-Komponenten) eine Benutzerschnittstelle bereitstellt bzw. eine Person standardmäßig erreichbar ist. Dies kann z.B. dafür genutzt werden, um die Bordcomputer-Bedienoberfläche (WH4.0-MMS) eines Harvesters (WH4.0-Komponente) oder die Standard-App (WH4.0-MMS) eines Forsteinrichters (WH4.0-Komponente) zu definieren.

Jedem WH4.0-Ding können *Endpoints* via *allEndpoints* zugeordnet werden, über die es mit dem jeweils ausgewählten Transportprotokoll (OPC UA, REST, MQTT, S³I-B …) erreichbar ist und kommunizieren kann. Dabei schränkt S³I die Auswahl der Protokolle nicht ein. Endpoints erlauben dann Zugriff auf die komplette Struktur oder einzelne Eigenschaften und Services des WH4.0-Dings. Teile dieser Endpoints können zusätzlich als Standard-Endpoints (*defaultEndpoints*) definiert werden, an die im Zweifelsfall Nachrichten geschickt werden sollen, um ggf. Mehrdeutigkeiten aufzulösen. Mehrere Standard-Endpoints sollten dabei unterschiedliche Protokolle aufweisen.

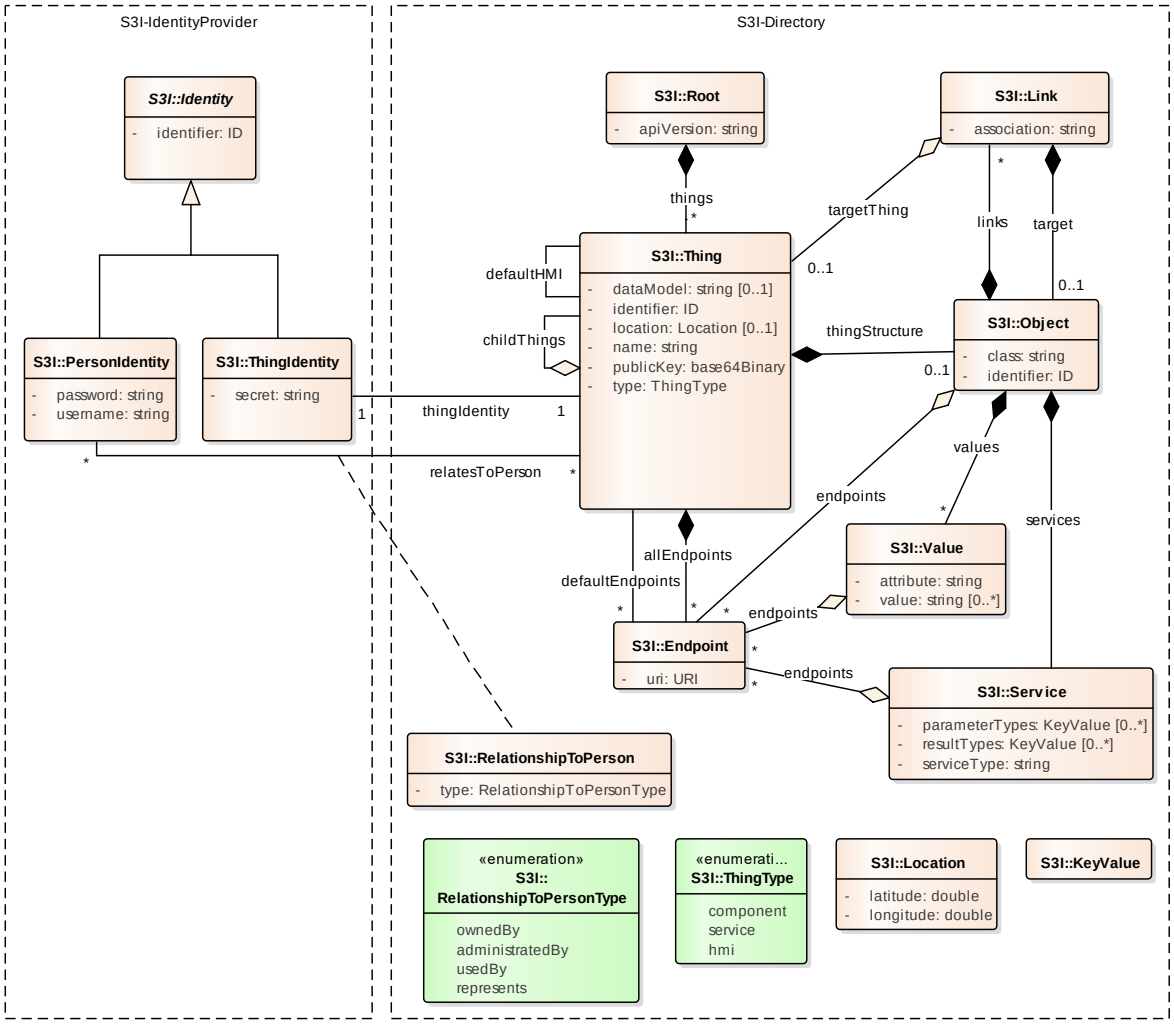

Abbildung 2-1: Das konzeptuelle Datenmodell des S³I-identityProviders (links) und des S³I-Directories (rechts)

Um dies näher zu spezifizieren, kann die Struktur des WH4.0-Dings (bzw. die relevanten Teile davon) bereits im S³I-Directory (via *thingStructure*) abgebildet werden. Dazu wird ein generisches objekt-orientiertes Datenmodell aus Objekten (*Object*), Werten (*Value*), bereitgestellten Diensten (*Service*) sowie Objektstrukturen über Verknüpfungen (*Link*) bereitgestellt. Durch die Generik dieses Datenmodells können darüber beliebige, konkret vom jeweiligen Ding verwendete Datenmodelle abgebildet werden. Die jeweils instanziierten Elemente des konkreten Zieldatenmodells werden über die Eigenschaften *Object::class*, *Value::attribute* sowie *Link::association* spezifiziert. Den Objekten, Werten und

Services können dann im Detail die Endpoints des WH4.0-Dings zugeordnet werden, um darüber auszudrücken, dass ein bestimmter Endpoint z.B. Zugriff auf ein Teilobjekt bzw. einen Teilobjektbaum bietet, einen ganz bestimmten Wert eines Assets bereitstellt (z.B. Drehzahl des Motors eines Harvesters) oder einen ganz bestimmten Service bereitstellt. Ein *Value* kann neben Endpoints auch einen gecachten Wert (*value*) enthalten. Ein *Service* kann sowohl für die Beschreibung eigenständiger WH4.0-Dienste als auch für die von einer WH4.0-Komponente über ihren DZ bereitgestellten Funktionen genutzt werden. Dabei ist der Typ des Service (*serviceType*) – auch im Sinne eines Namens – zu spezifizieren. Darüber hinaus können die notwendigen Eingangsparameter (*parameterTypes*) sowie Ergebnistypen (*resultTypes*) als Schlüssel-Wert-Paare (*KeyValue*) definiert werden. Der Schlüssel beschreibt dabei den Namen des Parameters bzw. Teilergebnisses, der Wert den zugehörigen Datentyp. Datentypen können in beiden Fällen sowohl einfache Datentypen (int, boolean, string …) als auch Dateiformate (als MIME-Typ) sein.

2.2.1 Abbildung auf JSON

Beispiele für S³I-Directory-Einträge und die ***Abbildung des konzeptuellen Datenmodells auf JSON*** für das Zielsystem Eclipse Ditto als Grundlage des S³I-Directory zeigen die folgenden Abschnitte. Allgemein werden in JSON Objekte auf JSON-Objekte (Maps) abgebildet. Objekteigenschaften werden zu Schlüssel-Wert-Paaren, mit dem Attributnamen als Schlüssel und dem Attributwert als Wert. Dasselbe gilt für Verknüpfungen (Assoziationen) mit anderen Objekten. Als Sonderfall werden Mengen von *KeyValue*-Instanzen (für *parameterTypes* und *resultTypes*) jeweils auf ein JSON-Objekt und damit eine Map der entsprechenden Schlüssel-Wert-Paare abgebildet. Spezifisch für Ditto wird der *Thing::identifier* auf die *thingId* abgebildet, weil diese dort eine besondere Semantik hat. In diesem Zuge wird eine *policyId* ergänzt, die identisch zur *thingId* gesetzt wird und für die Rechteverwaltung von Ditto genutzt wird. Alle IDs von *ThingIdentities* sind UUIDs mit dem Namensraum *s3i*, z.B. „s3i:ef39a0ae-1f4a-4393-9508-ad70a4d38a63“. Ebenso erfordert Ditto, dass sämtliche spezifischen Eigenschaften der WH4.0-Dinge unter *attributes* angeordnet werden. Insgesamt ergibt sich damit die Struktur gemäß der folgenden Bespiele.

2.2.2 Beispiel: WH4.0-Komponente Harvester

Im Folgenden ist der beispielhafte Eintrag einer WH4.0-Komponente eines Harvesters und seines DZ beschrieben.

```
{
    "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://..."]
                    }]
                }
            }]
        }
    }
}
```


Abbildung 2-2: Directory-Eintrag für eine WH4.0-Komponente Harvester im JSON-Format

Im einzelnen umfasst die Beschreibung der WH4.0-Komponente „Harvester“ in Abbildung 2-2:

- Eine *thingId* (=*identifier*) „s3i:ef39…“
- eine *policyId*
- alle spezifischen Eigenschaften gemäß dem konzeptuellen Datenmodell des S^{3}I-Directories unterhalb von *attributes*
- Die Angabe des Besitzers der Maschine (*ownedBy*) als *identifier* der zugehörigen *PersonIdentity* im S^{3}I-IdentityProvider „606d…“
- Den Anzeigename (*name*) „Harvester“
- Die räumliche Verortung (*location*) in Form von *longitude* / *latitude*
- Einen öffentlichen Schlüssel für die sichere Kommunikation (*publicKey*)
- Den Typ (*type*) des WH4.0-Dings, hier *component*
- Das vom WH4.0-Ding verwendete Datenmodell (*dataModel*) ForestML 4.0 („fml40“), mit dem die Struktur des DZ der WH4.0-Komponente beschrieben ist
- Die Liste aller Endpoints (*allEndpoints*) der WH4.0-Komponente Harvester, im Beispiel über S^{3}I-B, OPC UA und MQTT
- Einen Ausschnitt der hierarchischen Struktur (*thingStructure*) des DZ der WH4.0-Komponente Harvester im weiter oben definierten Datenmodell
    - Beginnend mit einem Wurzelobjekt (*Object*) der Klasse (*class*) „fml40.Harvester“

- und einer Service-Funktion (*services*) mit Namen (*serviceType*) „fml40.Harvester.getProductionData“, die Angabe des zugehörigen Endpoints (*endpoints*) „s3ib://s3i:ef39a0ae-1f4a-4393-9508-ad70a4d38a63“, der Spezifikation einer leeren Parameterliste (*parameterTypes*) sowie der Rückgabewerte (*resultTypes*) mit einem Wert „productionData“ vom Typ „application/stanford2010+hpr“
- und einer Verknüpfung (*links*) gemäß Assoziation (*association*) „fgml40.Harvester.engine“ und einem Zielobjekt (*target*)
- Zielobjekt (*Object*) der Klasse (*class*) „fml40.Engine“
- und einem Wert (*values*) zu Attribut (*attribute*) „fml40.Engine.rpm“ und Angabe des zugehörigen Endpoints (*endpoints*) „mqtt://…“, über den dieser Wert abrufbar ist

2.2.3 Beispiel: WH4.0-Dienst Berechnung Bestandesinventurattribute

Im Folgenden ist der beispielhafte Eintrag eines WH4.0-Dienstes zur Berechnung von Bestandesinventurattributen beschrieben.


```
{
    "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"}
            }
        ]}
    }
}
```


Abbildung 2-3: Directory-Eintrag für einen WH4.0-Dienst zur Berechnung von Bestandesinventurattributen

Der in Abbildung 2-3 beschriebene WH4.0-Dienst „Bestandesinventurattribute-Dienst“ umfasst im einzelnen:

- Eine *thingId* (=*identifier*) „s3i:ab07…“
- eine *policyId*
- alle spezifischen Eigenschaften gemäß dem konzeptuellen Datenmodell des S^{3}I-Directory unterhalb von *attributes*
- Die Angabe des Besitzers des WH4.0-Dienstes (*ownedBy*) als *identifier* der zugehörigen *PersonIdentity* im S^{3}I-IdentityProvider „606d…“

- Den Anzeigename (*name*) „Bestandesinventurattribute-Dienst“
- Den Typ (*type*) des WH4.0-Dings, hier *service*
- Einen öffentlichen Schlüssel für die sichere Kommunikation (*publicKey*)
- Die Liste aller Endpoints (*allEndpoints*) der WH4.0-Kompontente Harvester, im Beispiel über S^3I-B
- Einen Ausschnitt der (hier flachen) Struktur (*thingStructure*) des WH4.0-Dienstes
    - Beginnend mit einem typenlosen Wurzelobjekt (*Object*)
    - zwei verknüpften Service-Funktionen (*services*)
    - einmal vom Typ (*serviceType*) „calculateStock“
        - mit Endpunkt (*endpoints*) „s3ib://s3i:ab0717b0-e025-4344-8598-bcca1d3d5d47-1“
        - und Angabe der für den Aufruf notwendigen Parameter (*parameterTypes*)
        - konkret den Parameter „surface“ mit erwartetem Datentyp „application/zipped-shapefile“ (gezipptes ShapeFile) – d.h. im Sinne einer BASE64-kodierten Datei
        - sowie Angabe der Datentypen und Namen der Rückgabewerte (*resultTypes*)
        - konkret den Rückgabewert „stock“ vom Typ „double“
    - einmal vom Typ (*serviceType*) „calculateStandAttributes“
        - mit Endpunkt (*endpoints*) „s3ib://s3i:ab0717b0-e025-4344-8598-bcca1d3d5d47-2“
        - und Angabe der für den Aufruf notwendigen Parameter (*parameterTypes*)
        - konkret den Parameter „surface“ mit erwartetem Datentyp „application/zipped-shapefile“ (gezipptes ShapeFile) – d.h. im Sinne einer BASE64-kodierten Datei
        - und dem Parameter „referenceDate“ mit Datentyp „date“
        - sowie Angabe der Datentypen und Namen der Rückgabewerte (*resultTypes*)
        - konkret den Rückgabewert „standAttributes“ vom Typ „application/x-forestgml“ im Sinne einer BASE64-kodierten ForestGML-Datei

2.2.4 Beispiel: WH4.0-Komponente Forsteinrichter und seine WH4.0-MMS App

Im Folgenden ist der beispielhafte Eintrag einer WH4.0-Komponente des forstlichen Sachverständigen „Sachverstaendiger Schmitz“ (also einer Person) beschrieben.

```
{
    "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-a0b966flab01",
        "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"
                        }
                    ]
                }
            }]
        }
    }
}
```


Abbildung 2-4: Directory-Eintrag für eine WH4.0-Komponente forstlicher Sachverständiger im JSON-Format

Im einzelnen umfasst die Beschreibung der WH4.0-Komponente aus einem Sachverständigen und seinem DZ in Abbildung 2-4:

- Eine *thingId* (=*identifier*) „s3i:2a27…“
- eine *policyId*
- alle spezifischen Eigenschaften gemäß dem konzeptuellen Datenmodell des S³I-Directory unterhalb von *attributes*
- Die Angabe des Besitzers der WH4.0-Komponente (*ownedBy*) als *identifier* der zugehörigen *PersonIdentity* im S³I-IdentityProvider „800e…“
- Die Angabe der Person (*PersonIdentity*), die durch den DZ dieser WH4.0-Komponente repräsentiert wird (*represents*) als *identifier* der zugehörigen *PersonIdentity* des Sachverständigen im S³I-IdentityProvider „800e…“ (hier identisch zum Besitzer)
- Den Anzeigename (*name*) „Sachverständiger Schmitz“
- Den Typ (*type*) des WH4.0-Dings, hier *component*
- Das vom WH4.0-Ding verwendete Datenmodell (*dataModel*) ForestML 4.0 („fml40“), mit dem die Struktur des DZ der WH4.0-Komponente beschrieben ist
- Die Angabe, dass das WH4.0-Ding mit dem *identifier* „s3i:6f58…“ (also die App des Sachverständigen) die Standard-WH4.0-MMS (*defaultHMI*) von diesem WH4.0-Ding ist
- Die Liste aller Endpoints (*allEndpoints*) der WH4.0-Komponente, im Beispiel ist die Liste leer, weil der Sachverständige ausschließlich über seine App (WH4.0-MMS) kommunizieren kann / will

- Einen Ausschnitt der hierarchischen Struktur (*thingStructure*) des DZ der WH4.0-Komponente „Sachverständiger“ im weiter oben definierten Datenmodell
    - Beginnend mit einem Wurzelobjekt (*Object*) der Klasse (*class*) „fml40.Person“
    - einem Wert (*values*) zu Attribut (*attribute*) „fml40.Person.occupation“ und direkter Angabe des zugehörigen Wertes „consultant“ – ohne Angabe eines Endpoints, der Wert ist lediglich über das S³I-Directory verfügbar
    - und einer Verknüpfung (*links*) gemäß Assoziation (*association*) „fgml40.Person.address“ und einem Zielobjekt (*target*)
    - Zielobjekt (*Object*) der Klasse (*class*) „fml40.Address“
    - und einem Wert (*values*) zu Attribut (*attribute*) „fml40.Address.name“ und direkter Angabe des zugehörigen Wertes „Schmitz“ – ohne Angabe eines Endpoints, der Wert ist lediglich über das S³I-Directory verfügbar
    - und einem Wert (*values*) zu Attribut (*attribute*) „fml40.Address.city“ und direkter Angabe des zugehörigen Wertes „Arnsberg“ – ohne Angabe eines Endpoints, der Wert ist lediglich über das S³I-Directory verfügbar


```
{
    "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"]
    }
}
```


Abbildung 2-5: Directory-Eintrag für die WH4.0-MMS App des forstlichen Sachverständigen im JSON-Format

Die Beschreibung der WH4.0-MMS „Forsteinrichter-App“ umfasst:

- Eine *thingId* (=*identifier*) „s3i:6f58…“
- eine *policyId*
- alle spezifischen Eigenschaften gemäß dem konzeptuellen Datenmodell des S³I-Directory unterhalb von *attributes*
- Die Angabe des Besitzers der WH4.0-MMS (*ownedBy*) als *identifier* der entsprechenden *PersonIdentity* im S³I-IdentityProvider „800e…“ (hier der Sachverständige als Person)
- Die Angabe der Administratoren der WH4.0-MMS (*administratedBy*) als *identifier* der zugehörigen *PersonIdentity* im S³I-IdentityProvider „800e…“ (hier der Sachverständige als Person)
- Die Angabe der Nutzung der WH4.0-MMS (*usedBy*) als *identifier* der zugehörigen *PersonIdentity* im S³I-IdentityProvider „800e…“ (hier der Sachverständige als Person)
- Den Anzeigename (*name*) „App von Sachverständigem Schmitz“
- Den Typ (*type*) des WH4.0-Dings, hier *hmi*
- Einen öffentlichen Schlüssel für die sichere Kommunikation (*publicKey*)
- Die Liste aller Endpoints (*allEndpoints*) der WH4.0-MMS, im Beispiel ein S³I-B-Endpoint
- Die Liste der Standard-Endpoints (*defaultEndpoints*) der WH4.0-MMS, im Beispiel der eine S³I-B-Endpoint

2.3 S³I-B-Protokoll

Abbildung 2-6 zeigt das konzeptuelle Datenmodell zum Aufbau einer Nachricht (*Message*) für das S³I-B-Protokoll des S³I-Brokers. Eine Nachricht umfasst Angaben zum sendenden WH4.0-Ding (*sender*) in Form einer Identifikation aus dem S³I-Directory (*identifier* einer *Thing*-Instanz, siehe Abbildung 2-1) sowie einem Identifikator für die Nachricht (*identifier*). Jeder Nachricht können ein oder mehrere Empfänger zugeordnet werden (*receivers*), ebenso jeweils in Form von Identifikationen aus dem S³I-Directory.

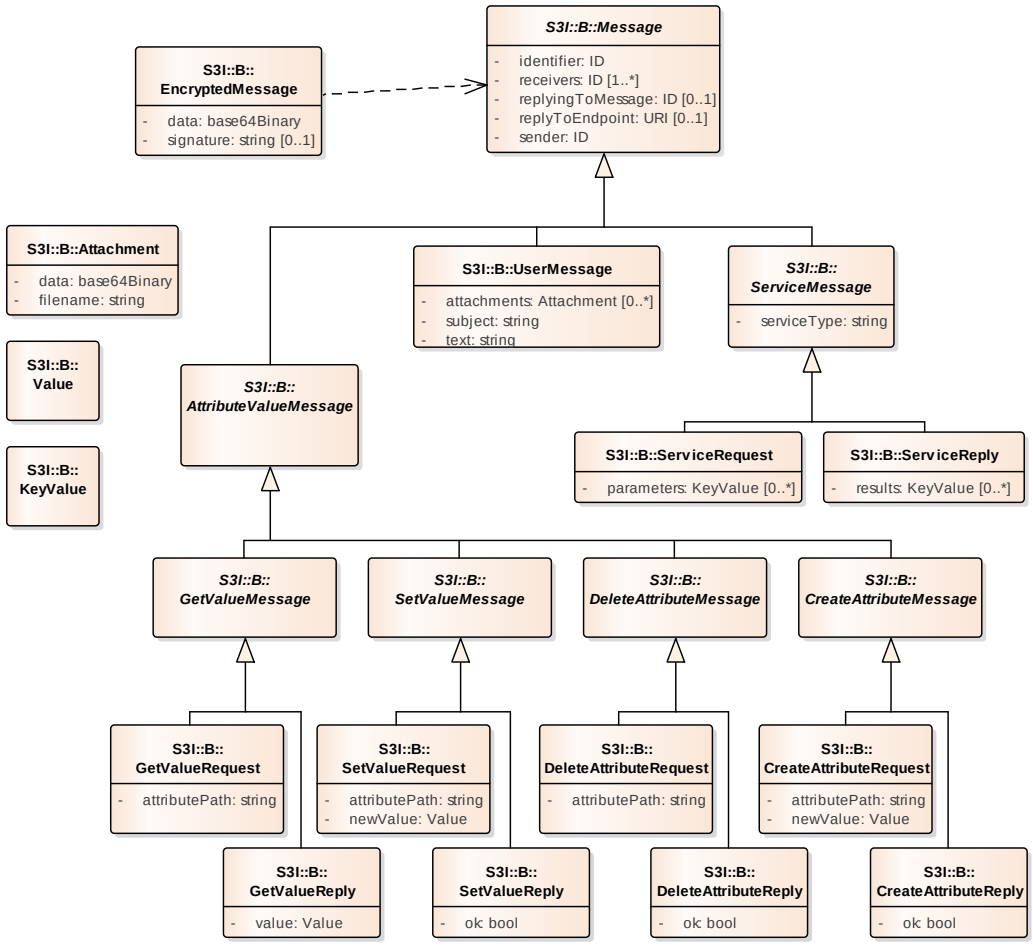

Abbildung 2-6: Das konzeptuelle Datenmodell für Nachrichten des S³I-B-Protokolls, das für den S³I-Broker genutzt wird

Weiterer Bestandteil jeder Nachricht kann die Information sein, ob sie die Antwort auf eine vorherige Nachricht darstellt (*replyingToMessage* mit deren *identifier*) oder an welchen Endpoint mögliche Antworten zurückgeschickt werden sollen (*replyToEndpoint*).

Nachrichten können in ihrer Gesamtheit verschlüsselt und signiert werden (*EncryptedMessage*, Details siehe Abschnitt 4).

Darüber hinaus gibt es drei grundlegende Ausprägungen von Nachrichten:

- Menschenlesbare Nachricht zwischen Nutzern (*UserMessage*)
- Maschinelesbare Nachricht für den Zugriff auf Service-Funktionen eines WH4.0-Dienstes oder einer WH4.0-Komponente (*ServiceMessage*)
    - zum Aufruf (*ServiceRequest*) und
    - für die zugehörige Antwort (*ServiceReply*)

- Maschinenlesbare Nachricht für den Zugriff auf Daten (*AttributeValueMessage*) des WH4.0-Dings
    - Zum Lesen von Attributwerten (*GetValueMessage*) mit Anfrage (*GetValueRequest*) und Antwort (*GetValueReply*)
    - Zum Schreiben von Attributwerten (*SetValueMessage*) mit Anfrage (*SetValueRequest*) und Antwort (*SetValueReply*)
    - Zum Löschen von Attributen (*DeleteAttributeMessage*) mit Anfrage (*DeleteAttribute-Request*) und Antwort (*DeleteAttributeReply*)
    - Zum Erzeugen von Attributen (*CreateAttributeMessage*) mit Anfrage (*CreateAttribute-Request*) und Antwort (*CreateAttributeReply*)

Eine *UserMessage* umfasst dann die üblichen Felder für E-Mail-artige Kommunikation für Betreff (*subject*), Textinhalt (*text*) und Dateianhänge (*Attachment*). Ein *Attachment* kombiniert Dateiname (*filename*) mit zugehörigen BASE64-kodierten Daten (*data*).

Die Angaben zu einer *ServiceMessage* korrespondieren zu den Angaben in Abschnitt 2.1 bzgl. der Spezifikation von WH4.0-Dienst-Schnittstellen im S³I-Directory. Allgemein wird über den *serviceType* beschrieben, welche WH4.0-Dienst-Funktion beim Empfänger genau aufgerufen werden soll. Ein Aufruf (*ServiceRequest*) enthält dann die notwendigen Aufruf-Parameter (*parameters*) in Form einer Liste von Schlüssel-Wert-Paaren mit dem Parameternamen als Schlüssel und Parameterwert als Wert. Analog enthält eine Service-Antwort (*ServiceReply*) die Ergebnisse (*results*) als Liste von Schlüssel-Wert-Paaren.

Über eine *AttributeValueMessage* kann auf Eigenschaften eines WH4.0-Dings (i.d.R. einer WH4.0-Komponente) zugegriffen werden. Dieser Zugriff auf die Eigenschaften des WH4.0-Dings (z.B. auf die Eigenschaften einer Forstmaschine durch einen auf der Forstmaschine selbst („Edge“) bereitgestellten Digitalen Zwilling) selbst ist nicht zu verwechseln mit einem Zugriff auf den Eintrag des WH4.0-Dings im S³I-Directory. Dazu sind jeweils verschiedene Parameter notwendig. In allen Varianten gibt der *attributePath* den Pfad zum betrachteten Attribut des WH4.0-Dings an. Der *value* bzw. *newValue* beschreibt den gelesenen bzw. neu zu schreibenden Wert dieses Attributs. Beim verändernden Zugriff gibt ein Flag (*ok*) den Erfolgsstatus der Aktion als Antwort zurück.

2.3.1 Abbildung auf JSON

Beispiele für S³I-B-Nachrichten sowie die ***Abbildung des konzeptuellen Datenmodells auf JSON*** zeigen die folgenden Abschnitte für eine Nachricht zwischen Personen (2.3.2), einen Aufruf eines WH4.0-Dienstes (2.3.3) und eine Antwort eines WH4.0-Dienstes (2.3.4) sowie die Abfrage eines Attributwertes einer WH4.0-Komponente (2.3.5) und die zugehörige Antwort (2.3.6). Inhalte sind dabei beispielhaft oder beispielhaft den eigentlichen Inhalt beschreibend (in spitzen Klammern).

Allgemein erfolgt die Abbildung des konzeptuellen Datenmodells auf JSON wie beim S³I-Directory. Wie beim S³I-Directory auch werden *KeyValue*-Listen direkt auf ein JSON-Objekt mit Schlüssel-Wert-Paaren darin abgebildet. Der Typ einer Nachricht – also *UserMessage*, *ServiceRequest* oder *ServiceReply*, *GetValueRequest* oder *GetValueReply* etc. – wird im JSON-Format über die Eigenschaft *messageType* abgebildet.

2.3.2 Beispiel: Nachricht zwischen Personen

Abbildung 2-7 zeigt eine Benutzernachricht von einem Waldbesitzer an seinen Sachverständigen – genauer zwischen deren jeweiligen WH4.0-MMS (z.B. ihren Apps). Im Einzelnen besteht die Nachricht aus:

- Der *thingId*/*identifier* des sendenden WH4.0-Dings, hier der WH4.0-MMS (App) des Waldbesitzers

- Die Kennung (*identifier*) der Nachricht selbst
- Die Liste der Empfänger (*receivers*) mit einem Empfänger (dem Sachverständigen) in Form der *thingId*/*identifier* seiner WH4.0-MMS (App) (*receiver*)
- Der Typ der Nachricht (*messageType*) – hier eine Benutzernachricht (*userMessage*)
- Die Angabe des Endpoints, an den der Sender (Waldbesitzer) die Nachricht erwartet (*replyToEndpoint*), hier der Endpoint seiner App
- Eine Liste von Dateianhängen (*attachments*) mit einer Datei „foto.jpg“ (*filename*) und ihrem BASE64-kodierten Inhalt (*data*)
- Der Betreff (*subject*)
- Der Text (*text*)


```
{
    "sender": "s3i:2aafd97c-ff05-42b6-8e4d-e492330ec959",
    "identifier": "s3i:1385e09e-3e93-4c2f-92d3-345698c40407",
    "receivers": ["s3i:6f58e045-fd30-496d-b519-a0b966f1ab01"],
    "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"
}
```


Abbildung 2-7: Beispielhafte S³I-B-Nachricht zwischen Personen (JSON-Format)

2.3.3 Beispiel: Service-Aufruf

Abbildung 2-8 zeigt eine beispielhafte Nachricht zum Aufruf eines WH4.0-Dienstes zur Vorratsberechnung durch eine WH4.0-MMS, im Beispiel der App eines Sachverständigen. Im Einzelnen besteht die Nachricht aus:

- Der *thingId*/*identifier* des sendenden WH4.0-Dings, hier der WH4.0-MMS (App) des Sachverständigen
- Die Kennung (*identifier*) der Nachricht selbst
- Die Liste der Empfänger (*receivers*) mit einem Empfänger (dem WH4.0-Dienst) in Form von dessen *thingId*/*identifier* (*receiver*)
- Der Typ der Nachricht (*messageType*) – hier ein Dienst-Aufruf (*serviceRequest*)
- Die Angabe des Endpoints, an den der Sender (Sachverständige) die Nachricht erwartet (*replyToEndpoint*), hier der Endpoint seiner App
- Die Angabe, welche Service-Funktion (*serviceType*) genau aufgerufen werden soll, hier „calculateStock“
- Die Angabe der dazu notwendigen Aufrufparameter (*parameters*), hier der Parameter „surface“ mit einem BASE64-kodierten gezippten 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": "..."}
}
```


Abbildung 2-8: Beispielhafte S³I-B-Nachricht zum Aufruf eines WH4.0-Dienstes (JSON-Format)

2.3.4 Beispiel: Service-Antwort

Abbildung 2-9 zeigt die Antwort des WH4.0-Dienstes zur Vorratsberechnung an den Sachverständigen (bzw. dessen App) aus dem vorherigen Abschnitt. Im Einzelnen besteht die Nachricht aus:

- Der *thingId*/*identifier* des sendenden WH4.0-Dings, hier der WH4.0-Dienst
- Die Kennung (*identifier*) der Nachricht selbst
- Die Liste der Empfänger (*receivers*) mit einem Empfänger – der WH4.0-MMS (App) des Sachverständigen – in Form von deren *thingId*/*identifier* (*receiver*)
- Der Typ der Nachricht (*messageType*) – hier eine Dienst-Anwort (*serviceReply*)
- Die Angabe, welche Service-Funktion (*serviceType*) genau aufgerufen wurde, hier „calculateStock“
- Die Angabe, auf welche Nachricht geantwortet wurde in Form von deren Kennung (*identifier*)
- Die Ergebnisse (*results*), hier das Ergebnis „stock“ mit Wert „123.4“ gemäß Spezifikation im S³I-Directory


```
{
    "sender": "s3i:ab0717b0-e025-4344-8598-bccald3d5d47",
    "identifier": "s3i:f13ef9dd-68d8-43c9-b920-99c943c42089",
    "receivers": ["s3i:6f58e045-fd30-496d-b519-a0b966flab01"],
    "messageType": "serviceReply",
    "serviceType": "calculateStock",
    "replyingToMessage": "s3i:08938da3-a2aa-47be-810c-9e6ee97e7fdb",
    "results": {"stock": "123.4"}
}
```


Abbildung 2-9: Beispielhafte S³I-B-Nachricht mit dem Ergebnis eines WH4.0-Dienstes (JSON-Format)

2.3.5 Beispiel: Attributwert-Anfrage an Digitalen Zwilling

Abbildung 2-10 zeigt ein Beispiel für die Anfrage eines Attributwertes. Der DZ eines Sägewerks fragt den DZ eines Forwarders nach dessen Eigenschaften („features“). Im Einzelnen besteht die Nachricht aus:

- Der *thingId*/*identifier* des sendenden WH4.0-Dings, hier der Digitale Zwilling des Sägewerks
- Die Kennung (*identifier*) der Nachricht selbst
- Die Liste der Empfänger (*receivers*) mit einem Empfänger – hier der DZ des Forwarders – in Form von dessen *thingId*/*identifier* (*receiver*)
- Der Typ der Nachricht (*messageType*) – hier die Anfrage eines Attributwerts (*getValueRequest*)
- Die Angabe des Endpoints, an den der Sender die Nachricht erwartet (*replyToEndpoint*), hier der S³I-B-Endpoint des DZ des Sägewerks
- Der Pfad zum relevanten Attribut, hier alle Eigenschaften gleichzeitig („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"
}
```


Abbildung 2-10: Beispielhafte S³I-B-Nachricht für den Abruf eines Attributwertes eines Forwarders (JSON-Format)

2.3.6 Beispiel: Attributwert-Antwort von Digitalem Zwilling

Abbildung 2-11 zeigt ein Beispiel für die Antwort des Digitalen Zwilling des Forwarders auf die vorherige Anfrage eines Attributwertes vom Sägewerk. Im Einzelnen besteht die Nachricht aus:

- Der *thingId*/*identifier* des sendenden WH4.0-Dings, hier der Forwarder
- Die Kennung (*identifier*) der Nachricht selbst
- Die Liste der Empfänger (*receivers*) mit einem Empfänger – hier das Sägewerk – in Form von deren *thingId*/*identifier* (*receiver*)
- Der Typ der Nachricht (*messageType*) – hier die Rückgabe eines Attributwerts (*getValueReply*)
- Die Angabe, auf welche Nachricht geantwortet wurde (*replyingToMessage*)
- Der Wert (*value*) des abgefragten Attributs, hier alle Eigenschaften („features“) des Forwarders, konkret die mit dem Forwarder verknüpfte Waage (verkürzt)


```
{
    "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"
            }]
        }]
    }]
}
```


Abbildung 2-11: Beispielhafte S³I-B-Nachricht des DZ Forwarder mit dem Ergebnis zum Abruf eines Attributwertes (JSON-Format)

2.4 S³I-Repository

Während das S³I-Directory ein Verzeichnis aller WH4.0-Dinge einschließlich der (optionalen) Beschreibung ihrer Struktur und ihrer Zugangspunkte bereitstellt, kann das ebenfalls auf Eclipse Ditto basierende S³I-Repository die Daten der WH4.0-Dinge selbst JSON-basiert aufnehmen. Analog zum S³I-Directory wird durch das Basissystem Ditto zunächst nur eine Grundstruktur aus *thingId* und *policyId* gefordert. Eigenschaften werden im Allgemeinen unter den Schlüsseln *attributes* (für eher statische) bzw. *features* (für eher variable) erwartet. Abbildung 2-12 zeigt diese Grundstruktur.


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


Abbildung 2-12: Grundlegende Struktur eines Eintrags im S³I-Repository

Für die Modellierung von WH4.0-Dingen entwickelt das KWH4.0 das Datenmodell ***ForestML 4.0***, das in einem folgenden KWH4.0-Standpunkt näher beschrieben wird. Wie im S³I-Directory wird die gemäß des ForestML 4.0-Datenmodells aufgebaute, hierarchische Beschreibung eines WH4.0-Dings vollständig unterhalb des Schlüssels *attributes* geführt, um eine geschlossene Abbildung zu ermöglichen. Das Ditto-Basissystem bietet jedoch eine optimierte Datenhaltung und einen optimierten Zugriff auf sich häufig ändernde Eigenschaften unterhalb des Schlüssels *features*. Daher empfiehlt das KWH4.0 eine

optionale Auslagerung solcher Werte in den *features* Bereich, die am Ursprungsort dann über ein spezielles Schlüsselwort „*ditto-feature:idX*“ gekennzeichnet wird.

Wichtig ist festzustellen, dass ein und dasselbe konzeptuelle Datenmodell – für WH4.0 empfiehlt das KWH4.0 ForestML 4.0 – leicht ***unterschiedlich*** auf die JSON-Strukturen im S³I-Directory bzw. im S³I-Repository abgebildet werden:

- Im ***S³I-Directory*** erfolgt unterhalb des Schlüssels *thingStructure* eine eher ***indirekte Abbildung*** über das in Abbildung 2-1 definierte Metamodell aus *S3I::Object*, *S3I::Link*, *S3I::Value* und *S3I::Service*. Dadurch können zunächst beliebige Datenmodelle (nicht nur ForestML 4.0) im S³I-Directory abgebildet werden. Zudem können durch diesen Ansatz auch die notwendigen Metadaten des S³I-Directory insbesondere in Form der Endpunkte (S3I::Endpoint) mit den entsprechenden Strukturen verknüpft werden. Beispielsweise kann so einem Attributwert eine Endpoint-Adresse zugeordnet werden, über die der Wert abrufbar ist. Ebenso kann so die Signatur einer Service-Schnittstelle definiert werden.
- Im ***S³I-Repository*** erfolgt eine eher ***direkte Abbildung*** des (ForestML 4.0-) Datenmodells auf JSON. Das heißt insbesondere aus Attributen und Assoziationen im Datenmodell werden unmittelbar Schlüssel im JSON-Dokument. Service-Funktionen werden hingegen gar nicht abgebildet, weil sie keine Daten beschreiben und bereits über die Datenmodell-Beschreibung der zugehörigen Klassen abgebildet sind. Objekte selbst werden hingegen identisch als JSON-Objekt mit Angabe der Klasse (*class*) spezifiziert.

2.4.1 Beispiel: Radlader

Abbildung 2-13 zeigt einen Ausschnitt eines in ForestML 4.0 modellierten Radladers, der im S³I-Repository gespeichert ist.

- Er wird über seine *thingId* s3i:7f654f13-11ac-413d-841c-10a8e431fc35 identifiziert. Dies ist dieselbe ID wie sie auch im S³I-Directory und im S³I-IdentityProvider für dieses WH4.0-Ding genutzt wird.
- Die Zugriffsrechte werden über die Policy mit *policyId* s3i:7f654f13-11ac-413d-841c-10a8e431fc35 spezifiziert.
- Die komplette hierarchische ForestML 4.0-konforme Beschreibung des WH4.0-Dings liegt unterhalb des Schlüssel *attributes* (Block „1“ im Bild). Zu beachten ist dabei der Unterschied zwischen dem durch das Ditto-Basissystem definierten Schlüssel *features* und dem gleichnamigen, durch ForestML 4.0 definierten Schlüssel *features* (innerhalb Block „1“ im Bild), die eine unterschiedliche Bedeutung aufweisen. Näheres dazu im oben angekündigten KWH4.0-Standpunkt zu ForestML 4.0.
- Für die optimierte Datenverwaltung und den optimierten Zugriff durch das Basissystem Ditto können einzelne Eigenschaften in Ditto-Features ausgelagert werden (Block „2“ im Bild). Im Beispiel ist dies insbesondere die Eigenschaft *currentLevel* des Tanks des Radladers (Block „1“ im Bild). Der Wert dieser Eigenschaft wird über das Schlüsselwort „*ditto-feature:id3*“ in ein Ditto-Feature mit der Kennung „id3“ ausgelagert (Block „2“ im Bild). Hinweis: Die anderen beiden ausgelagerten Features („id1“ und „id2“) werden vom ausgeblendeten Teil des Beispiels („[9 lines]“) referenziert.

Abbildung 2-13: ForestML 4.0-konform modellierter Radlader im S³I-Repository (1: ForestML 4.0-spezifischer Teil; 2: ausgelagerte Ditto-Features für optimierte Datenverwaltung)

3 Autorisierung im S³I-Directory

Details zur Autorisierung im S³I-Directory in S³I folgen in der nächsten Version dieses Standpunkts.

4 Verschlüsselung und Signierung im S³I-B-Protokoll

Zur ***Verschlüsselung*** und Signierung von Nachrichten wird im S³I-B-Protokoll das Verfahren ***Pretty Good Privacy (PGP)*** (Abbildung 4-1) genutzt. Zur Kennzeichnung von S³I-B-Endpunkten, die verschlüsselte (und auch unverschlüsselte) Nachrichten unterstützen, wird das URI Schema „s3ib**s**://“ genutzt. Endpunkte, die nur unverschlüsselte Nachrichten unterstützen, sollen das URI Schema „s3ib://“ verwenden.

PGP ist ein Verschlüsselungsstandard, der kryptographischen Datenschutz und Authentifizierung für die Datenkommunikation bietet. Die Network Working Group beschreibt das ***Open PGP Message Format*** in ***RFC 4880***[10]. Das Format verwendet eine Kombination aus symmetrischen und asymmetrischen Schlüsseln.

Bei der Kryptographie mit symmetrischem Schlüssel wird eine Nachricht mit denselben kryptographischen Schlüsseln ver- und entschlüsselt. Die beiden Schlüssel können unterschiedliche Formate haben, sie basieren jedoch auf demselben „Geheimnis". Dieses Geheimnis muss zwischen den beteiligten Parteien geteilt werden, was einen sicheren Kanal für den Austausch des Geheimnisses voraussetzt. Die asymmetrische Kryptographie vermeidet das gemeinsame Geheimnis. Jeder Benutzer stellt ein Schlüsselpaar zur Verfügung, das aus einem öffentlichen und einem privaten Schlüssel besteht. Daten, die mit einem dieser Schlüssel verschlüsselt sind, können nur mit dem entsprechenden Gegenstück entschlüsselt werden. Unter der Voraussetzung, dass der private Schlüssel immer privat gehalten wird, bewirkt eine Verschlüsselung von Daten mit dem öffentlichen Schlüssel, dass nur der Inhaber des privaten Schlüssels diese Daten entschlüsseln kann. Auf diese Weise wird bei Verwendung des öffentlichen Schlüssels einer Person sichergestellt, dass nur sie die Nachricht entschlüsseln kann, solange die Zugehörigkeit des privaten Schlüssels zum erwarteten Empfänger gewahrt bleibt. Auf diese Weise wird die Vertraulichkeit einer PGP-Nachricht gewährleistet.

Der private Schlüssel kann nicht nur zur Entschlüsselung, sondern auch zum ***Signieren*** von Nachrichten verwendet werden. Auch hier gilt, dass nur der erwartete Besitzer des privaten Schlüssels den privaten Schlüssel verwendet, so dass man, wenn Daten mit diesem Schlüssel signiert werden, sicher sein kann, dass dieser Besitzer die Nachricht signiert hat. Die Signatur einer Nachricht ist ein Hash[11], der mit dem privaten Schlüssel verschlüsselt wird. Sie wird an die eigentliche Nachricht angehängt. Wenn der Empfänger die Signatur mit dem jeweiligen öffentlichen Schlüssel des Versenders entschlüsselt, sollte sie mit demjenigen Hash der Nachricht übereinstimmen, den er selbst berechnet hat. Wenn die Nachricht geändert wird, würde die Verifizierung fehlschlagen, da die Signatur nicht auf der geänderten Nachricht, sondern auf der ursprünglichen basiert, und mit dem Hash der veränderten Nachricht verglichen wird. Die Verwendung solcher Signaturen gewährleistet die Authentizität („Kommt die Nachricht auch wirklich vom angegebenen Absender?") und Integrität der Nachricht.

Im Vergleich zwischen der Kryptographie mit symmetrischem und asymmetrischem Schlüssel ist die symmetrische Verschlüsselung leichter, was bedeutet, dass sie weniger Rechenressourcen benötigt und dadurch im Vergleich zur asymmetrischen Verschlüsselung schneller ist. Dennoch bietet sie keine Authentifizierung der Nachricht durch Signaturen und ist problematisch hinsichtlich der Verteilung der Schlüssel. PGP kombiniert die Vorteile der asymmetrischen und symmetrischen Verschlüsselung, indem es die Daten mit einem symmetrischen Schlüssel verschlüsselt, diesen Schlüssel mit einem Public-Key-Kryptographie-Ansatz verschlüsselt und zusammen mit den Daten versendet. Auf diese Weise wird die relativ ressourcen- und zeitaufwändige Verschlüsselung mit öffentlichem Schlüssel nur auf den symmetrischen Schlüssel angewendet und die eigentliche Datenverschlüsselung erfolgt schneller und ressourcenschonender mit einem symmetrischen Schlüssel. Der Prozess ist in Abbildung 4-1 dargestellt. Gibt es mehrere Empfänger, muss der symmetrische Schlüssel für jeden Empfänger mit seinem jeweiligen öffentlichen Schlüssel verschlüsselt werden. Die Nachricht kann auch vor der Verschlüsselung signiert werden.

[10] https://tools.ietf.org/html/rfc4880
[11] Ein Hash ist hier im Sinne einer Prüfsumme zu verstehen.

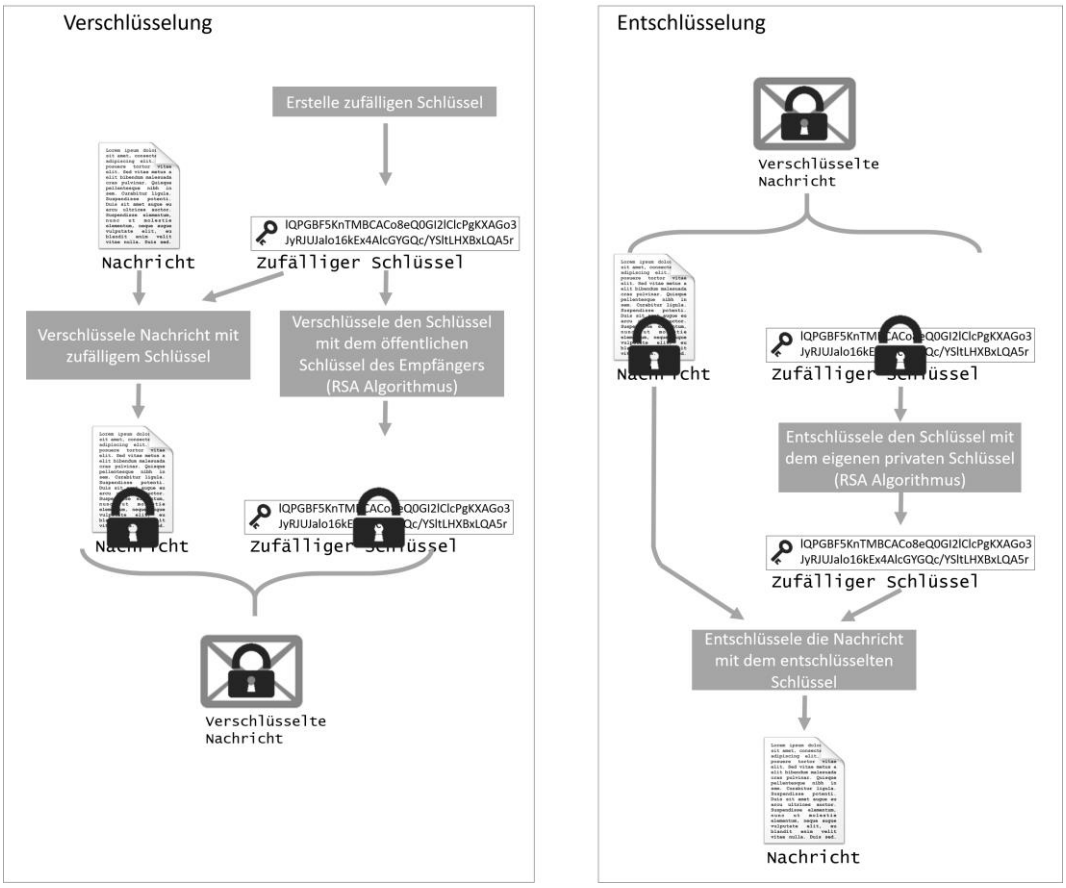

Abbildung 4-1: Verschlüsselung und Entschlüsselung von S3I-B-Nachrichten mit dem PGP-Verfahren[12]

Der RFC-Standard schreibt keinen konkreten Verschlüsselungsalgorithmus vor, sondern gibt an, welche Algorithmen von einer PGP-Anwendung implementiert werden müssen, sollen und können. In S^{3}I soll der RSA (Rivest-Shamir-Adleman)-Algorithmus mit einem 2048 Bit langen Schlüssel für die Verschlüsselung mit öffentlichem Schlüssel und ein AES (Advanced Encryption Standard) mit einem 512 Bit langen Schlüssel für die symmetrische Verschlüsselung verwendet werden.

5 Authentifizierung mit dem S^{3}I-IdentityProvider

Im Kontext von S^{3}I wird der ***OpenID Connect (OIDC)***[13] Standard zur Authentifizierung genutzt. Konkret kommen dabei zwei „Login Flows“ zur Verwendung.

5.1 Person und WH4.0-MMS

Soll sich eine ***Person über eine WH4.0-MMS*** authentifizieren, so wird der „Authorization Code Flow“[14] genutzt. Vereinfacht gesprochen meldet sich die Person dazu mit ihrem Benutzernamen und Passwort (gemäß ihrer *PersonIdentity*) an der WH4.0-MMS an. Die WH4.0-MMS authentifiziert sich dann mit diesem Benutzernamen und Passwort (der *PersonIdentity*) sowie ihrem eigenen Geheimnis (ihrer *ThingIdentity*) gegenüber dem S^{3}I-IdentityProvider und erhält dafür ein JWT. Das JWT kann dann zur Authentifizierung der Kombination aus Person und WH4.0-MMS gegenüber Dritten genutzt werden.

[12] Abbildung basiert auf *„How PGP encryption works“* (https://en.wikipedia.org/wiki/Pretty_Good_Privacy#/media/File:PGP_diagram.svg) von xaedes&jfreax&Acdx. Lizenz: CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0).
[13] https://openid.net/connect/
[14] https://auth0.com/docs/flows/concepts/auth-code

5.2 WH4.0-Ding ohne Person

Soll hingegen ein ***WH4.0-Ding selbstständig*** ohne direkte Beteiligung einer Person tätig werden, so wird der „Client Credentials Flow“[15] genutzt. Hier authentifiziert sich das WH4.0-Ding nur mit seinem Geheimnis (seiner *ThingIdentity*) gegenüber dem S³I-IdentityProvider und erhält dafür ein JWT. Das JWT kann dann zur Authentifizierung des WH4.0-Dings gegenüber Dritten genutzt werden.

6 Anwendungsfälle

Im Folgenden sind drei beispielhafte Anwendungsfälle für das Zusammenspiel der S³I-Dienste beschrieben. Diese können auch mit den beiliegenden Code-Beispielen nachgestellt werden.

6.1 Nachrichtenaustausch zwischen Personen

Ein Waldbesitzer schickt seinem forstlichen Sachverständigen eine vertrauliche Nachricht.

- Der Waldbesitzer gibt seine Anmeldedaten, also Benutzername und Passwort seiner persönlichen Identität (*PersonIdentity*), in eine WH4.0-MMS ein – als Beispiel eine App
    - Die App authentifiziert sich gegenüber dem S³I-IdentityProvider und erhält ein JWT gemäß Abschnitt 5.1
- Der Waldbesitzer öffnet in seiner App einen Dialog zum Schreiben sicherer S³I-B-Nachrichten und sucht nach dem Nachnamen seines Sachverständigen
    - Die App sucht im S³I-Directory nach dem eingegebenen Nachnamen und findet die zugehörige WH4.0-Komponente des Sachverständigen. Hierzu authentifiziert sich die App mit dem JWT beim S³I-Directory.
    - Da diese WH4.0-Komponente des Sachverständigen über keinen eigenen Endpoint verfügt, fragt die App das S³I-Directory nach der Standard-WH4.0-MMS (*defaultHMI*) dieser WH4.0-Komponente. Im Beispiel zeigt dieser Eintrag auf eine App des Sachverständigen.
    - Von dieser WH4.0-MMS wählt sie den Standard-S³I-B-Endpoint (aus *defaultEndpoints*).
- Der Waldbesitzer verfasst eine Nachricht mit Betreff und Text und hängt ein Foto als Dateianlage an
- Der Waldbesitzer drückt auf Senden
    - Die App bildet eine JSON-basierte S³I-B-Benutzernachricht gemäß Abschnitt 2.3.2
        - Mit *sender* als *identifier* der App des Waldbesitzers (eine WH4.0-MMS) und *receiver* als *identifier* der App des Sachverständigen (auch eine WH4.0-MMS)
        - Unter *replyToEndpoint* trägt sie ihren eigenen S³I-B-Endpoint ein, damit Antworten direkt an die App zurückgeschickt (falls es sich nicht um die Standard-WH4.0-MMS oder deren Standard-Endpoint halten sollte)
    - Die App signiert und verschlüsselt die Nachricht gemäß Abschnitt 4
    - Die App authentifiziert sich mit dem JWT am S³I-Broker
    - Die App übergibt die verschlüsselte Nachricht über das AMQP-Protokoll an die in der URI[16] hinterlegte Queue des S³I-Brokers
- Der Sachverständige erhält eine Benachrichtigung von der WH4.0-konformen App auf seinem Smartphone über eine neue Nachricht
    - Die App authentifiziert sich gegenüber dem S³I-IdentityProvider und erhält ein JWT gemäß Abschnitt 5.1
    - Die App authentifiziert sich mit dem JWT gegenüber dem S³I-Broker

[15] https://auth0.com/docs/flows/concepts/client-credentials
[16] Uniform Resource Identifier; https://tools.ietf.org/html/rfc3986

  - Der S³I-Broker benachrichtigt dazu die WH4.0-MMS des Sachverständigen, im Beispiel auch eine App, über die neue Nachricht in der Queue
- Der Sachverständige ruft die neue Nachricht ab und liest sie
  - Die App ruft die neue Nachricht aus ihrer Queue ab
  - Die App prüft die Signierung des Nachrichteninhalts und entschlüsselt ihn gemäß Abschnitt 4

6.2 Aufruf eines Dienstes zur Vorratsberechnung

Ein Sachverständiger will für einen Umring den Vorrat über Fernerkundungsmethoden von einem WH4.0-Dienst berechnen lassen.

- Der Sachverständige nutzt dazu eine GIS-fähige App als WH4.0-MMS
- Der Sachverständige gibt seine Anmeldedaten, also Benutzername und Passwort seiner persönlichen Identität (*PersonIdentity*), in die App ein
  - Die App authentifiziert sich gegenüber dem S³I-IdentityProvider und erhält ein JWT gemäß Abschnitt 5.1
- Der Sachverständige zeichnet den gewünschten Umring und ruft für diesen eine Funktion für den Dienstaufruf in seiner App auf
  - Die App authentifiziert sich mit ihrem JWT beim S³I-Directory
  - Die App sucht im S³I-Directory nach WH4.0-Diensten mit *"serviceType": "calculateStock"*
  - Die App findet genau einen solchen WH4.0-Dienst
  - Die App ruft die Parameter-Beschreibung (*parameterTypes*) des zugehörigen *Service*-Eintrags ab und erkennt daran, dass der (einzige) Parameter *surface* als gezipptes ShapeFile übermittelt werden soll (*"surface": "application/zipped-shapefile"*) und stellt den Umring entsprechend bereit
  - Die App ruft die URI des Standard-S³I-B-Endpoints dieses *Service*-Eintrags ab (aus *defaultEndpoints*)
  - Die App bildet eine JSON-basierte S³I-B-Nachricht mit einem Service-Aufruf gemäß Abschnitt 2.3.3
    - Mit *sender* als *identifier* der WH4.0-MMS (App) und *receiver* als *identifier* des WH4.0-Dienstes
    - Mit der Angabe ihres S³I-B-Endpoints für die Antwort (*replyToEndpoint*)
  - Die App signiert und verschlüsselt die Nachricht gemäß Abschnitt 4
  - Die App authentifiziert sich mit ihrem JWT am S³I-Broker
  - Die App übergibt die verschlüsselte Nachricht über das AMQP-Protokoll an die in der URI hinterlegte Queue des S³I-Brokers
- Vorab einmalig bei seinem Start: Der WH4.0-Dienst zur Vorratsberechnung …
  - authentifiziert sich gegenüber dem S³I-IdentityProvider und erhält ein JWT gemäß Abschnitt 5.2
  - verbindet sich mit dem S³I-Broker,
  - authentifiziert sich bei ihm mit seinem JWT
- Im Betrieb beim Eintreffen der Nachricht von der App des Sachverständigen: Der WH4.0-Dienst zur Vorratsberechnung …
  - wird vom S³I-Broker über die neue Nachricht in seiner Queue informiert
  - ruft die neue Nachricht aus seiner Queue beim S³I-Broker ab
  - prüft die Signierung und entschlüsselt die Nachricht gemäß Abschnitt 4
  - parst den Service-Aufruf (*ServiceRequest*) und liest die Parameterdaten (*parameters*) aus

- führt seine Funktionalität aus (Vorratsberechnung)
- packt das Ergebnis (*results*) als Fließkommazahl (*"stock": 123.4*) in eine JSON-basierte S³I-B-Nachricht mit einem Service-Ergebnis (*ServiceReply*) gemäß Abschnitt 2.3.4
- signiert und verschlüsselt die Nachricht gemäß Abschnitt 4
- liest die URI für die Antwort aus dem Feld *replyToEndpoint* der eingehenden Nachricht aus
- übergibt die Nachricht an die in dieser URI hinterlegte Queue des S³I-Brokers

- Der Sachverständige erhält eine Benachrichtigung von seiner App, ruft diese ab und bekommt das Ergebnis angezeigt
  - Der Broker benachrichtigt die App des Sachverständigen über die neue Nachricht in der Queue
  - Die App ruft die neue Nachricht aus ihrer Queue ab
  - Die App prüft die Signierung des Nachrichteninhalts und entschlüsselt ihn gemäß Abschnitt 4
  - Die App identifiziert die Nachricht als Ergebnis (*ServiceReply*) eines vorherigen Service-Aufrufs und zeigt das Ergebnis (*results*) der beauftragten Vorrats-Berechnung an

6.3 Abruf einer Eigenschaft einer Forstmaschine

Ein Sägewerk (bzw. der DZ der WH4.0-Komponente Sägewerk) benötigt einen StanForD 2010-Datensatz von einem Harvester (bzw. vom DZ der WH4.0-Komponente Harvester), der dazu einen entsprechenden Service (als Funktionalität des DZ) bereitstellt.

- Der DZ Sägewerk …
  - authentifiziert sich gegenüber dem S³I-IdentityProvider und erhält ein JWT gemäß Abschnitt 5.2
  - authentifiziert sich mit dem JWT gegenüber dem S³I-Directory
  - greift auf den Eintrag des ihm bereits per *identifier* bekannten DZ Harvesters zu und sucht in dessen Struktur (*thingStructure*) nach einem *Service*-Eintrag mit *"serviceType": "fml40.Harvester.getProductionData"* mit leerer Parameterliste (*parameterTypes*) und Ergebnistyp (*resultTypes*) *"productionData": "application/stanford2010+hpr"*
  - ruft die URI des (bzw. eines) S³I-B-Endpoints dieses Service (*Service::endpoints*) ab
  - bildet eine JSON-basierte S³I-B-Nachricht mit einem Service-Aufruf (*ServiceRequest*) gemäß Abschnitt 2.3.3
    - Mit *sender* als *identifier* der WH4.0-Komponente Sägewerk und *receiver* als *identifier* der WH4.0-Komponente Harvester
    - Mit der Angabe seines S³I-B-Endpoints für die Antwort (*replyToEndpoint*)
  - signiert und verschlüsselt die Nachricht gemäß Abschnitt 4
  - authentifiziert sich mit dem JWT am S³I-Broker
  - übergibt die verschlüsselte Nachricht an die in der URI hinterlegte Queue des Brokers
- Der DZ Harvester …
  - authentifiziert sich gegenüber dem S³I-IdentityProvider und erhält ein JWT gemäß Abschnitt 5.2
  - verbindet sich mit dem S³I-Broker und authentifiziert sich dort mit dem JWT
  - wird vom S³I-Broker über die neue Nachricht in seiner Queue informiert
  - ruft die neue Nachricht aus seiner Queue beim S³I-Broker ab
  - prüft die Signierung und entschlüsselt die Nachricht gemäß Abschnitt 4
  - parst den Service-Aufruf (*ServiceRequest*) und liest die Parameterdaten (*parameters*) aus

  - führt die gewählte Funktionalität (*serviceType*) aus (stellt StanForD 2010 Produktionsdaten als HPR-Datei zusammen)
  - packt das Ergebnis (*results* mit *"productionData": "<BASE64-kodierter HPR-Dateiinhalt>"*) in eine JSON-basierte S³I-B-Nachricht mit einem Service-Ergebnis (*ServiceReply*) gemäß Abschnitt 2.3.4
  - signiert und verschlüsselt die Nachricht gemäß Abschnitt 4
  - liest die URI für die Antwort aus dem Feld *replyToEndpoint* der eingehenden Nachricht aus
  - übergibt die verschlüsselte Nachricht an die in der URI hinterlegte Queue des S³I-Brokers
- Der DZ Sägewerk …
  - wird vom S³I-Broker benachrichtigt
  - ruft die neue Nachricht aus seiner Queue ab
  - prüft die Signierung des Nachrichteninhalts und entschlüsselt ihn gemäß Abschnitt 4
  - parst die Nachricht (*ServiceReply*) und ordnet das Ergebnis (*results*) intern dem zugehörigen Auftrag zu (via *replyingToMessage*)