Forest Modeling Language 4.0

Konzeption und Einsatz der Forest Modeling Language 4.0 (fml40) zur Modellierung von Wald und Holz 4.0-Dingen

Ein KWH4.0-Standpunkt

07.04.2020

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

Förderhinweis

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

Version	Datum	Seiten	Änderungen
1.0	07.04.2020	Alle	Erste Version

Forest Modeling Language 4.0

Die Grundlage der Digitalisierung im Rahmen von Wald und Holz 4.0 (WH4.0) ist die ***Vernetzung*** aller „Dinge“. WH4.0-Dinge sind der Oberbegriff für WH4.0-Komponenten (Cyber-Physische Systeme wie Maschine, Geräte aber auch Menschen mit ihrem Digitalen Zwilling), WH4.0-Dienste (Softwaredienste) und WH4.0-Mensch-Maschine-Schnittstellen (auch WH4.0-MMS; mit dem Menschen kommunizierende Softwareprogramme auf dem PC, dem Smartphone oder als Webapplikation in der Cloud). Für eine erfolgreiche Vernetzung müssen die WH4.0-Dinge ***kommunizieren*** können. Dazu reicht aber die Spezifikation kompatibler Verbindungstechnik (z.B. WLAN, LoRa, 5G …) und darüber realisierten Kommunikationsprotokollen (OPC UA, S3I-B, REST …) allein nicht aus. Auch die ***Semantik*** der Kommunikation, die dabei „gesprochene Sprache“, muss untereinander verstanden werden. In Wald und Holz 4.0 entspricht dies einem ***einheitlichen Datenmodell für alle WH4.0-Dinge***, entsprechend dem Digitale Zwillinge, Dienste und MMS aufgebaut sind und entsprechend dem alle WH4.0-Dinge kommunizieren, so dass deren Struktur sowie verfügbaren Eigenschaften und Servicefunktionen einheitlich definiert sind und dadurch auch eine für alle definierte und verständliche Semantik haben.

Dieses ***konzeptuelle Datenmodell*** ist aber mehr als eine „Schnittstelle“ oder eine „Sprache“ für die Kommunikation untereinander. Viel mehr beschreibt es formal den ***Aufbau und Inhalt der WH4.0-Dinge*** selbst. Davon entkoppelt zu betrachten ist der ***Zugriff*** auf diese Inhalte über entsprechende ***Schnittstellen***. Diese können ebenso auf konzeptioneller Ebene Zugriffe wie das Lesen von Eigenschaften oder das Ausführen einer Servicefunktion in einer ***konzeptionellen Schnittstelle*** definieren. Diese konzeptuelle Schnittstelle wird in einem späteren KWH4.0-Standpunkt näher erläutert werden. Sie kann auf verschiedene Arten technisch umgesetzt werden, z.B. über OPC UA, S3I-B oder REST. Ebenso gibt es verschiedene Möglichkeiten der ***technischen Umsetzung des konzeptuellen Datenmodells für das Datenmanagement und für die Implementierung der Servicefunktionen***. Ein Beispiel ist das S3I-Repository für das Datenmanagement oder Python-Code für die Umsetzung der Servicefunktionen.

Das KWH4.0 hat die ***Forest Modeling Language 4.0*** (***ForestML 4.0*** oder kurz ***fml40***) als Vorschlag für ein solches Datenmodell entwickelt, das in diesem KWH4.0-Standpunkt vorgestellt wird. Die Forest Modeling Language 4.0 zählt zur forstlichen Sprachfamilie ***Forest Modeling Language (ForestML)***[1] und erweitert diese um eine Sprache zur Beschreibung von WH4.0-Dingen.

Im Rahmen dieses Standpunkts werden zunächst die Begriffe des ***konzeptuellen***, des ***logischen*** sowie des ***physikalischen Datenmodells*** anhand von ForestML erklärt. Es werden der grundlegende Aufbau der Sprache und ihre grundlegenden Modellierungskonzepte beschrieben. Darauf aufbauend werden ***Rollenkatalog*** und ***Featurekatalog*** (in ihrem aktuellen Entwicklungsstand) vorgestellt. Es folgen verschiedene Beispiele für die Modellierung von WH4.0-Dingen mittels fml40. Abschließend wird die Abbildung des konzeptuellen fml40-Datenmodells auf das ***JSON***-Format als logisches bzw. physikalisches Datenmodell beschrieben.

1 Konzeptuelle Datenmodelle

Ein ***konzeptuelles Datenmodell*** beschreibt die Konzepte und Ideen auf abstrakter Ebene. Dabei liegt der Fokus auf der Problem-Domäne und nicht auf der technischen Umsetzung. Das konzeptionelle Datenmodell gibt es nur genau einmal. Ein Beispiel ist das konzeptuelle Datenmodell der Sprachfamilie

[1] Siehe Umsetzungsstrategie Wald und Holz 4.0

ForestML (Abbildung 1-1). Für eine Formalisierung der Repräsentation werden Modellierungssprachen wie die Unified Modeling Language (UML) oder Entity-Relationship-Diagramme (ERD) verwendet.

Aus dem konzeptionellen können verschiedene ***logische Datenmodelle*** für verschiedene Umsetzungsvarianten abgeleitet werden. Dabei werden weiterhin dieselben Konzepte beschrieben, nur für unterschiedliche Lösungsansätze bzw. Zielsysteme. Beispiele sind die verschiedenen Ausprägungen von ForestML in OPC UA, JSON, in GML (hier dann als ForestGML) oder für relationale Datenbanken. Auf dieser Ebene können weiterhin allgemeine Modellierungssprachen wie UML oder ERD (mit um Implementierungsdetails angereicherten Modellen) oder domänenspezifische Sprachen wie SQL verwendet werden.

Aus logischen können wiederum ***physikalische Datenmodelle*** abgeleitet werden. Hierbei findet der Übergang in konkrete Softwaresysteme, Dateiformate oder Datenbanksysteme statt. Beispiele sind ForestML im XML-Format für OPC UA-Informationsmodelle („Nodesets“), ForestML im Ditto-konformen JSON-Format für das S3I-Repository, eine XML Schema Description (XSD) Datei für ForestGML oder eine konkrete Oracle-Instanz mit Tabellen für ForestML.

Insgesamt folgt daraus, dass ein einheitliches Datenmodell auf konzeptueller Ebene in den verschiedensten Zielsystemen unterschiedlich und damit bedarfsgerecht umgesetzt werden kann. Hierdurch wird insbesondere auch der Übergang zwischen unterschiedlichen logischen und physikalischen Modellen ermöglicht, die auf ein- und demselben konzeptuellen Datenmodell beruhen (Modelltransformation).

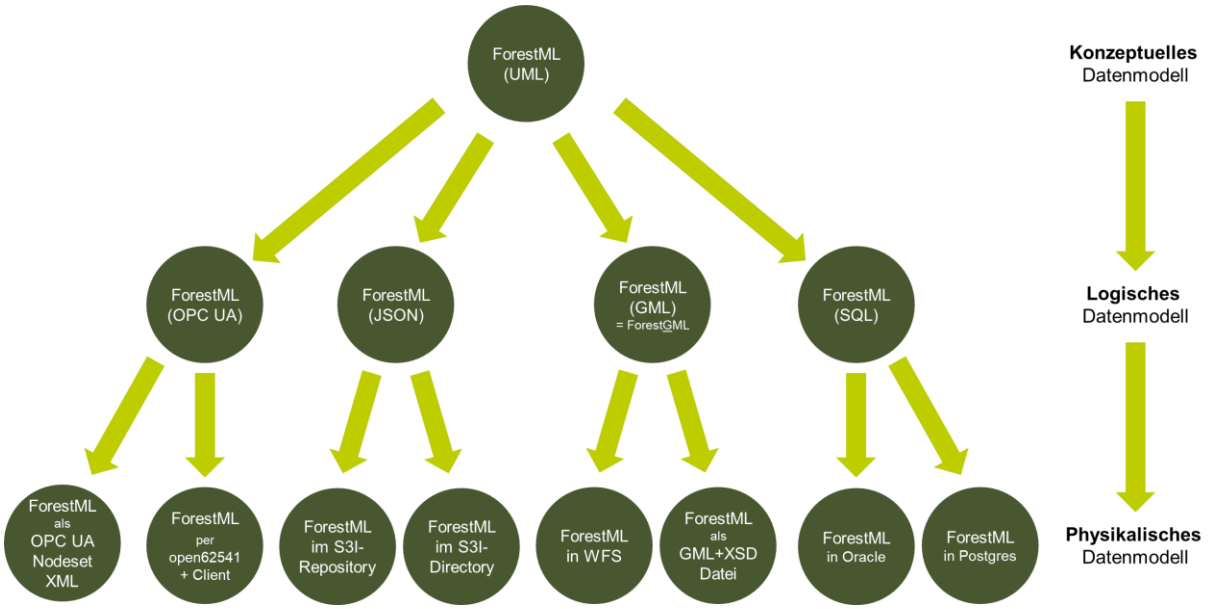

Abbildung 1-1: Veranschaulichung des Unterschieds zwischen konzeptuellen, logischen und physikalischen Datenmodellen am Beispiel ForestML

2 Grundlegender Aufbau

Abbildung 2-1 zeigt zunächst den grundlegenden Aufbau der Sprache. Die Grundidee ist zunächst eine Aufteilung in verschiedene Teilsprachen. In der Basissprache ***Modeling Language 4.0 (ml40)*** werden zunächst Forst-unspezifische, allgemeine Strukturen für die Modellierung von WH4.0-Dingen definiert. Darauf aufbauend folgen Forst-spezifische Erweiterungen in der Sprache ***Forest Modeling Language 4.0 (fml40)***, die für alle WH4.0-Nutzer gemeinsame, forstspezifische Strukturen definiert. Zur Modellierung von Detailinhalten werden dabei weitere Sprachen wie die Teilsprachen von ***ForestML (fml)*** zur

Waldbeschreibung, ***ELDATsmart*** zur Beschreibung des Holzhandels und -transports oder ***StanForD2010*** für Maschinendaten genutzt. Die Sprache kann zudem – wo notwendig – um weitere Strukturen (z.B. für hersteller-/betriebsinterne Besonderheiten) erweitert werden (***fml40<x>***, ***fml40<y>*** - z.B. fml40JohnDeere, fml40UPM …). Um eine Fragmentierung von Wald und Holz 4.0 zu vermeiden, sollten dabei möglichst nur sehr spezifische Erweiterungen umgesetzt und soweit möglich stattdessen allgemeine Strukturen von fml40 verwendet werden. ***Diese allgemeinen Strukturen von fml40 müssen hierzu in einem noch zu definierenden Prozess bedarfsgetrieben weiterentwickelt werden.*** Konkret geschieht dies gerade im Rahmen der sogenannten Praxisprojekte des KWH4.0.

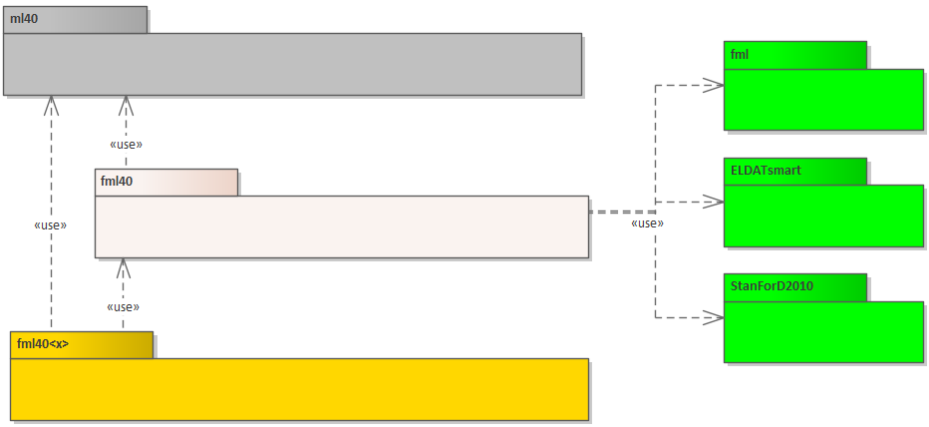

Abbildung 2-1: Aufteilung der Forest Modeling Language 4.0

Die für die Pakete genutzten Farben werden in den weiteren Diagrammen auch zur Unterscheidung der Sprachzugehörigkeit genutzt.

3 Grundlegende Modellierungskonzepte

Abbildung 3-1 zeigt die zentralen Klassen und gleichzeitig die grundlegenden Modellierungskonzepte der (Forst-unspezifischen) ***Modeling Language 4.0*** und damit auch ihrer (Forst-spezifischen) Erweiterung ***Forest Modeling Language 4.0***.

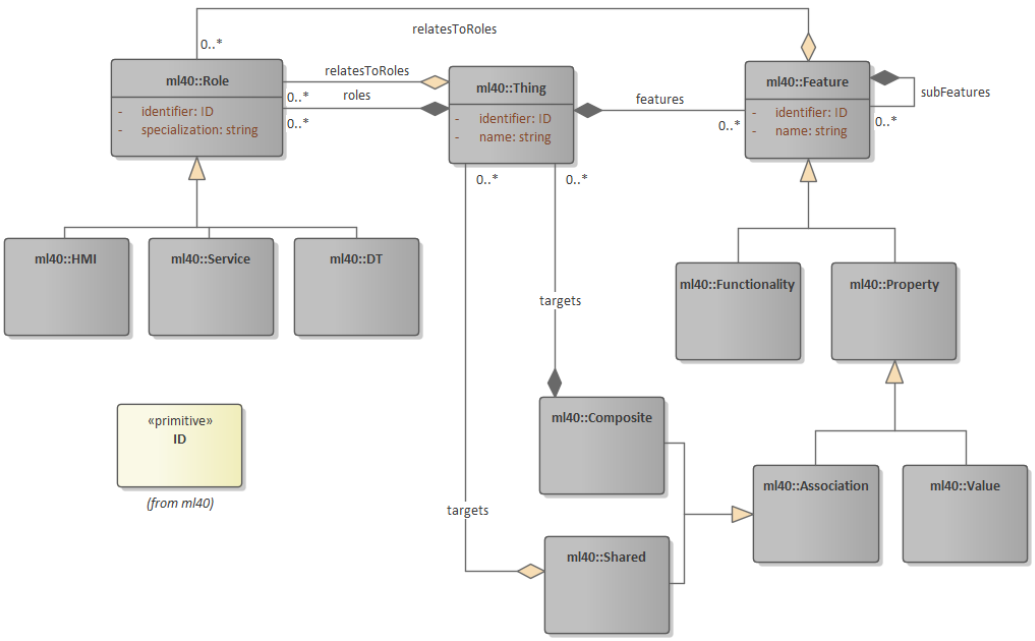

Abbildung 3-1: Grundlegende Struktur der Modeling Language 4.0

Zentrales Element ist das WH4.0-Ding (***ml40::Thing***), das eine eigenständige Einheit im Internet of Things (IoT) repräsentiert. Es setzt sich aus zwei wesentlichen Teilen zusammen: Rollen (***ml40::Role***) und Features (***ml40::Feature***).

Eine ***Rolle*** beschreibt die Aufgaben oder den Typ eines WH4.0-Dings. Über eine Vererbungshierarchie sind verschiedene Arten von Rollen definiert. Die grundlegenden sind zunächst Digitaler Zwilling (***ml40::DT***) einer WH4.0-Komponente, Dienst (***ml40::Service***) und Mensch-Maschine-Schnittstelle (MMS) (***ml40::HMI***). Darunter folgen spezifischere Rollen. WH4.0-Dinge können dabei mehrere Rollen innehaben (via ***roles***). So kann eine Person gleichzeitig Waldarbeiter und Waldbesitzer sein oder irgendwann einmal ein kombinierter Harvester-Forwarder entwickelt werden.

Der über ml40 und fml40 definierte ***Rollenkatalog***, also die Menge aller in (f)ml40 definierten Klassen von Rollen (siehe Abschnitt 4), kann auf verschiedene Weisen erweitert werden. Zunächst können über die Vererbung weitere Rollen definiert werden, so wie es bereits in fml40 gegenüber ml40 geschieht. Bei allgemeingültigen Rollen kann dies direkt in einer neuen Version von ml40 oder fml40 erfolgen. Handelt es sich um anwendungs- oder herstellerspezifische Rollen, so können die abgeleiteten Rollen auch in ein separates Erweiterungspaket (vgl. ***fml40<x>*** in Abschnitt 2) ausgelagert werden. Ein einfacherer Mechanismus, der keine Datenmodellanpassung benötigt ist die Spezialisierung einer vorhandenen Rolle über das Feld (***ml40::Role::specialization***). Darüber kann ein bestehender Rollentyp bei der Instanziierung an einem konkreten Ding noch einmal in seiner Bedeutung konkretisiert werden. Dabei gilt allgemein, dass immer auch Basisklassenrollen genutzt werden können, um WH4.0-Dinge zu spezifizieren. D.h. einem WH4.0-Ding darf auch z.B. eine allgemeine Rolle (z.B. ***fml40::ForestMachine***) zugeordnet werden, wenn es nicht näher spezifiziert werden kann oder soll.

Auf der anderen Seite werden WH4.0-Dinge durch ***Features*** (***ml40::Feature***), also Gestaltungsmerkmale, näher beschrieben. Die Grundidee ist, dass man flexibel diejenigen Features zuordnet, die für ein konkretes Ding notwendig sind. Hat ein WH4.0-Ding mehrere Rollen, so können einzelne Features im Bedarfsfall explizit einzelnen Rollen zugeordnet werden (***relatesToRoles***). Features können auch hierarchisch strukturiert aufgebaut werden (***subFeatures***). Features unterscheiden sich zunächst in solche zur Beschreibung von Funktionalität (***ml40::Functionality***) und solche zur Beschreibung von (strukturellen) Eigenschaften (***ml40::Property***). Über Functionality werden Bausteine aus Servicefunktionen z.B. zur Berechnung von Bestandesinventurattributen oder zur Abgabe von Produktionsdaten abgebildet. Zu bemerken ist dabei, dass ***Functionality***s mit Servicefunktionen sowohl von Digitalen Zwillingen von WH4.0-Komponenten, von eigenständigen WH4.0-Diensten sowie von WH4.0-MMS bereitgestellt werden können. Die Properties untergliedern sich zunächst weiter in solche zur Beschreibung konkreter Werte (***ml40::Value***) und solche zur Referenzierung anderer WH4.0-Dinge über Assoziationen (***ml40::Association***). Zu den Values zählen auch Container-Objekte z.B. für Adressen, die geographische Lage oder die Größendimensionen einer Maschine. Über Associations können die referenzierten WH4.0-Dinge entweder dem referenzierenden WH4.0-Ding untergeordnet (***ml40::Composite***, „Teil von“) oder diesem nur (in dem Sinne schwächer) zugeordnet werden (***ml40::Aggregate***). Durch letzteres können andere komplett eigenständig verwaltete „***top-level WH4.0-Dinge***“ referenziert werden.

Der ***Featurekatalog*** (siehe Abschnitt 5) bildet die Menge aller in (f)ml40 definierten Functionality- und Value-Klassen ab. Auch diese Kataloge können erweitert werden. Eine Möglichkeit ist auch hier die Vererbung innerhalb oder außerhalb von (f)ml40. Ohne Anpassung des Datenmodells lassen sich zudem einfache Eigenschaften im Sinne von Schlüssel-Wert-Paaren über die ***ml40::GenericProperty*** (siehe Abbildung 4-1) ergänzen.

Insgesamt erlaubt die Trennung in Rollen und Features das flexible Zusammensetzen von Dingen je nach Bedarf und Anwendungsfall. Dabei orientieren sich die Modellierungskonzepte der Forest Modeling Language 4.0 an den generischen Modellierungsansätzen der ***IEC 62264 / DIN EN 62264 „Integration von Unternehmens-EDV und Leitsystemen“***.

4 Rollenkatalog

Abbildung 4-1 zeigt den Rollenkatalog (aktueller Entwurfsstand) von ml40 und fml40. Nach der bereits o.g. grundlegenden Einteilung in Digitaler Zwilling (***ml40::DT***), Dienst (***ml40::Service***) und MMS (***ml40::HMI***) folgt eine Spezialisierung.

Über Rollen kann die Art eines WH4.0-Dings identifiziert werden. Dies kann z.B. bei der Suche „Alle Waldarbeiter in Region X“ oder im Szenario „Harvester will mit seinem benachbarten Forwarder reden“ genutzt werden. Dabei sind weitere Eigenschaften notwendig (in den beiden Beispielen die räumliche Lage), die über Values (Abschnitt 5) beschrieben werden.

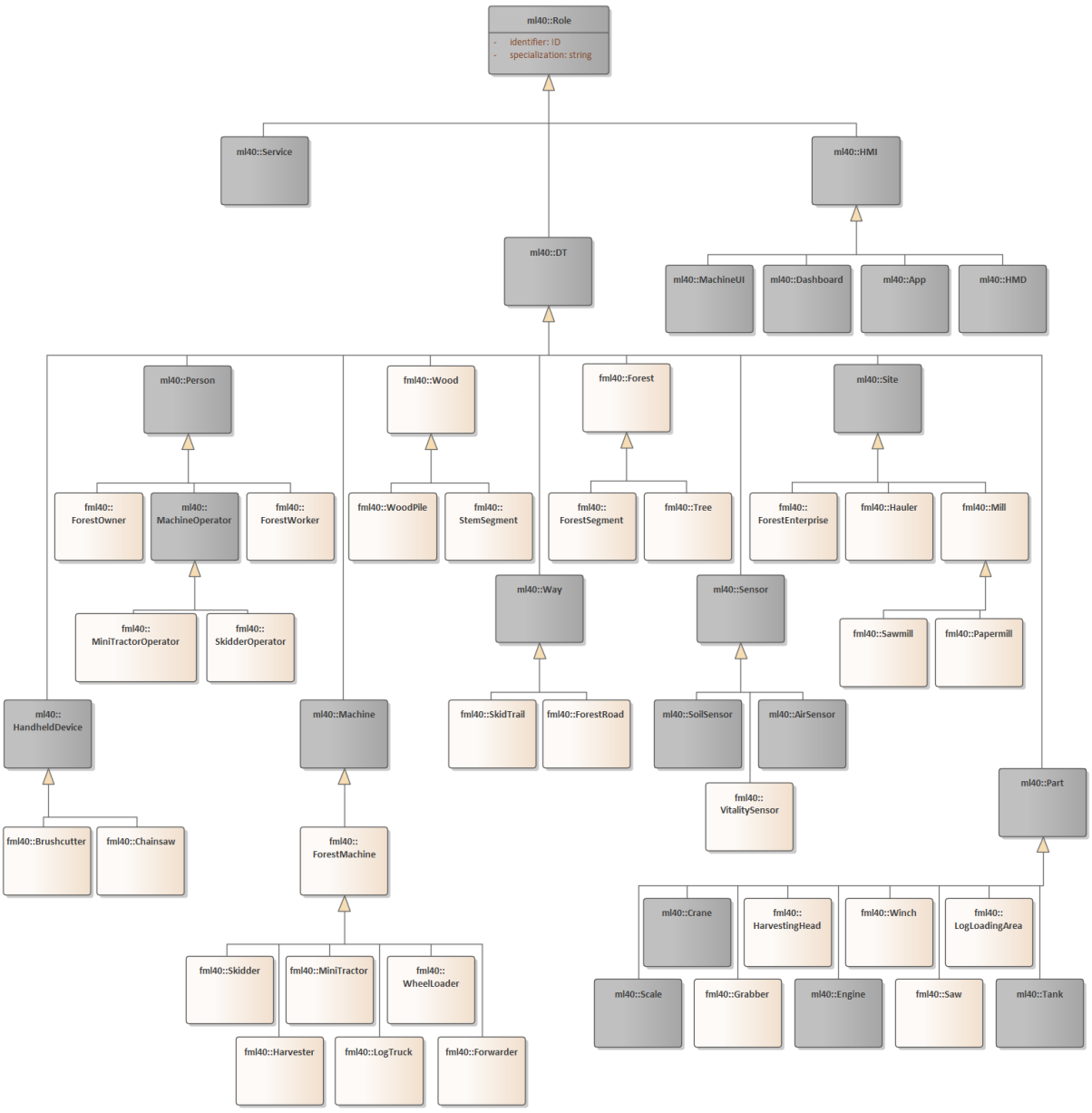

Abbildung 4-1: Rollenkatalog von (f)ml40

Bei den Diensten (***ml40::Service***) wird nicht weiter unterschieden, weil sich deren konkrete Aufgaben bereits durch die angebotenen Functionalitys definieren. Bei den MMS (***ml40::HMI***) gibt es aktuell Maschinen-Bediensysteme (User Interfaces (UI)) (***ml40::MachineUI***) z.B. für die Bordcomputer in der Harvesterkabine, Dashboards (***ml40::Dashboard***), Head-Mounted Displays (***ml40:HMD***) und „normale“ Anwendersoftware (***ml40::App***).

Bei den Digitalen Zwillingen (***ml40::DT***) gibt es derzeit die umfangreichste Rollensammlung, die von Maschinen (***ml40::Machine***) bzw. Forstmaschinen (***fml40::ForestMachine***) und handgeführten Geräten (***ml40::HandheldDevice***) über Personen (***ml40::Person***), Wald (***fml40::Forest***) und Holz (***fml40::Wood***) sowie Betriebsniederlassungen (***ml40::Site***) reicht.

5 Featurekatalog

Abbildung 5-1, Abbildung 5-2 und Abbildung 5-3 zeigen den aktuellen Entwicklungsstand des Featurekatalogs von (f)ml40. Insbesondere dieser Katalog befindet sich derzeit noch im Aufbau und wird sukzessive erweitert und ausdetailliert. In diesem Zusammenhang wird auch eine nähere Erläuterung der einzelnen Featureklassen, insbesondere der verschiedenen Functionalitys, ergänzt werden.

Der Eintrag ***fml::AbstractInventory*** beim Value ***fml40::InventoryData*** ist dabei eine Bezugnahme auf Inventurdaten im ForestML-Format.

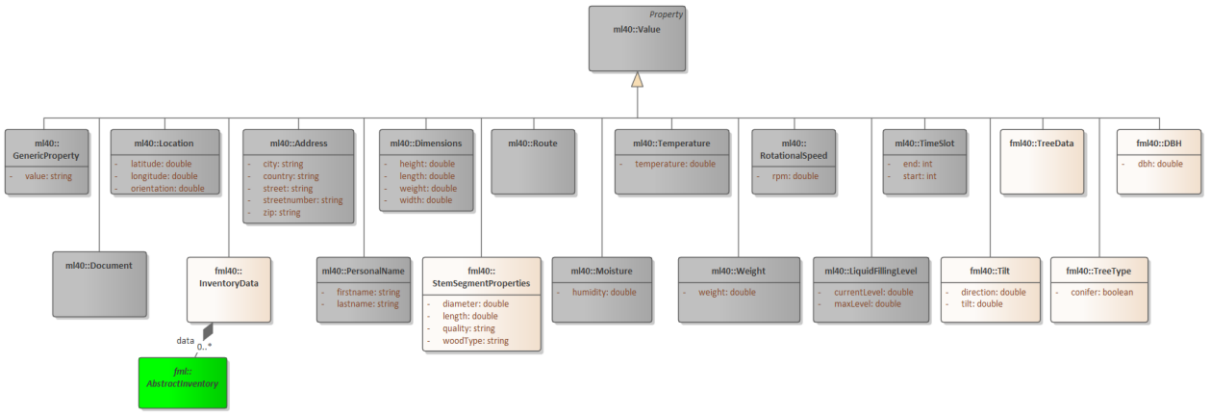

Abbildung 5-1: Featurekatalog von (f)ml40 – Werteigenschaften

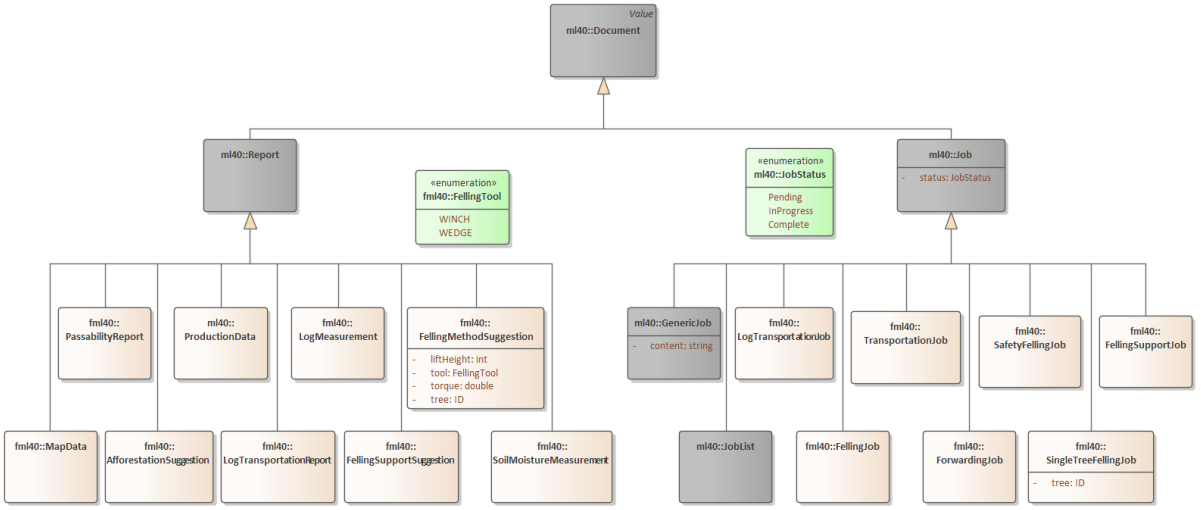

Abbildung 5-2: : Featurekatalog von (f)ml40 – Werteigenschaften – Dokumente

*Abbildung 5-3: Featurekatalog von (f)ml40 – Servicefunktionen (die Basisklasse **Functionality** wird aus Gründen der Übersichtlichkeit nicht explizit gezeigt)*

6 Beispiele

Der folgende Abschnitt zeigt einige Beispiele für konkrete WH4.0-Dinge auf Basis von fml40 in Form von UML-Objektdiagrammen[2]. Es ist zu beachten, dass es sich lediglich um erste Beispiele handelt, da sowohl Rollen- und Featurekatalog noch im Aufbau sind und da die WH4.0-Dinge mit unterschiedlichem Detailgrad modelliert werden können.

6.1 Fernerkundungsdienst

Abbildung 6-1 zeigt das Beispiel eines Fernerkundungsdienstes

- mit der allgemeinen Rolle Service (***ml40::Service***)
- mit zwei ***Functionality***s für die bereitgestellten Teil-Dienste zur Baumartenklassifikation (***fml40::ClassifiesTreeSpecies***) und zur Berechnung von Bestandesinventurattributen (***fml40::EvaluatesStandAttributes***). Die diesen ***Functionality***s zugeordneten Servicefunktionen wie z.B. ***calculateTreeSpeciesClassification(…)*** werden dabei in den entsprechenden Objektklassen (Abbildung 5-3) spezifiziert. Durch die Modellierung des WH4.0-Dienstes mit diesen beiden Functionalitys wird also deklariert, dass er die zugehörigen Servicefunktionen anbietet. Daher reicht es aus, im S3I-Directory nur die Functionalitys (und die Values) anzugeben, aber nicht deren einzelne Servicefunktionen (bzw. deren einzelne Attribute). Details siehe Abschnitt 7. Die technische Umsetzung des WH4.0-Dings muss dann eine entsprechende Implementierung der so deklarierten Servicefunktionen bereitstellen. Das gilt nicht nur für WH4.0-Dienste wie hier im Beispiel, sondern auch für Digitale Zwillinge von WH4.0-Komponenten oder WH4.0-MMS wie im Folgenden dargestellt.

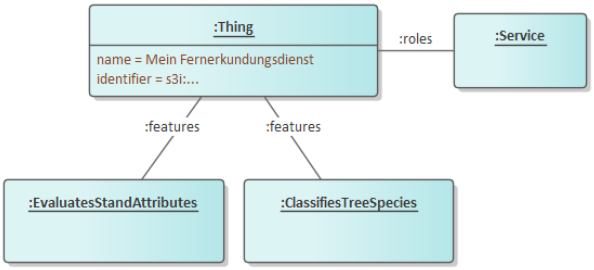

Abbildung 6-1: Beispiel für einen Fernerkundungsdienst in fml40

6.2 Harvester

Abbildung 6-2 zeigt das Beispiel eines Harvesters:

- mit der Rolle Harvester (***fml40::Harvester***)
- mit ***Functionality***s für die Annahme von Fällaufträgen (***fml40::AcceptsFellingJobs***), die Ausführung von Fällaufträgen (***fml40::Harvests***), die Beauftragung anderer Assets, insbesondere Forwardern (***ml40::ManagesJobs***), die Bereitstellung von Produktionsdaten (***fml40::ProvidesProducitonData***) und die Annahme von Näherungsalarmen (***fml40::AcceptsProximityAlert***)
- mit einem ***Value*** für seine aktuelle Position (***ml40::Location***)
- mit einer per ***ml40::Shared*** verbundenen Winde (als eigenständiges („top-level“) ***ml40::Thing*** mit Rolle ***fml40::Winch***), der für die konkrete Verwendung aber noch Features zugeordnet werden müssen (z.B. für das Einfahren der Winde (Functionality) oder Eigenschaften wie der maximalen Länge (Value))

[2] In UML-Objektdiagrammen stellen die Kästen konkrete Instanzen von Klassen, also Objekte, dar. Als Name wird die Klasse des Objekts mit einem „:“ davor genutzt und der Name wird insgesamt unterstrichen.

- mit per ***ml40::Composite*** verbundenem Bordcomputer (***ml40::Thing*** mit Rolle ***ml40::MachineUI***), Motor (***ml40::Thing*** mit Rolle ***ml40::Engine*** mit Eigenschaft ***ml40::RotationalSpeed***) und einem Kran (***ml40::Thing*** mit Rolle ***ml40::Crane***) mit damit wiederum per ***ml40::Composite*** verbundenem Harvesterkopf (***fml40::Thing*** mit Rolle ***ml40::HarvestingHead***). Letzterer kann über die exemplarische Functionality ***fml40::Grabs*** angesteuert werden. Insgesamt müssen aber auch diesen untergeordneten Dingen für deren konkrete Verwendung noch weitere Features zugeordnet werden

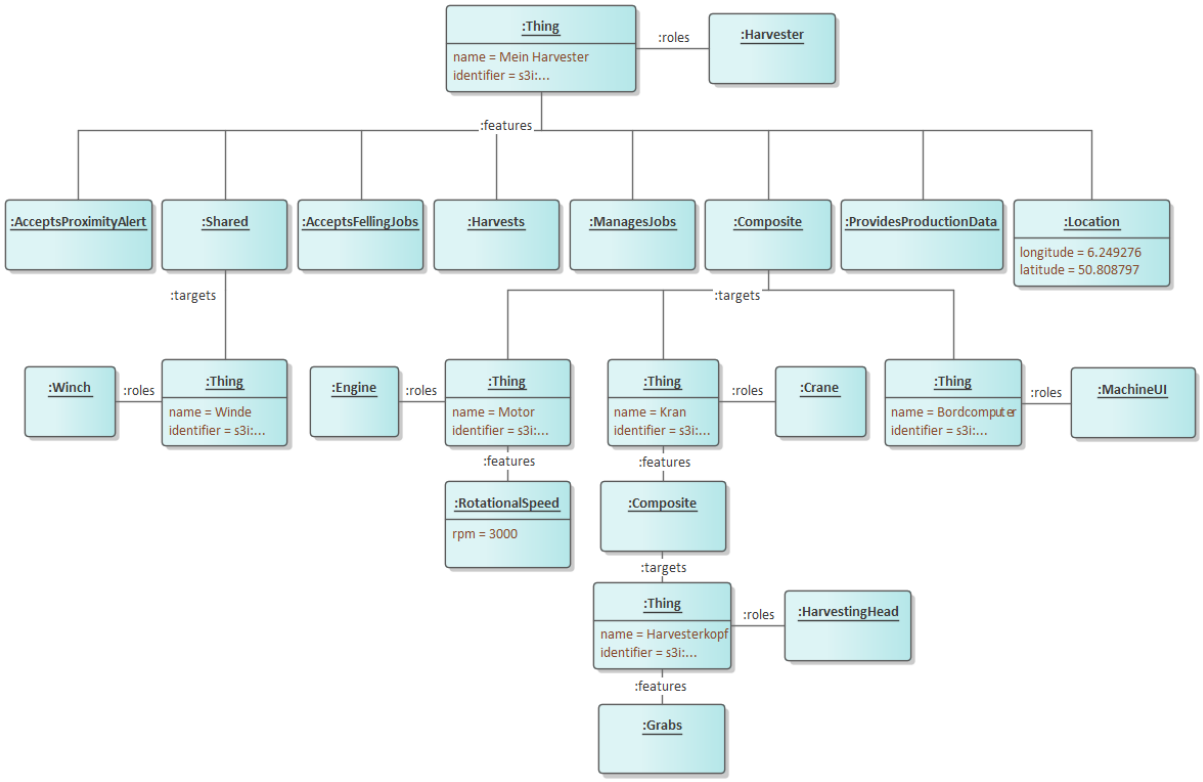

Abbildung 6-2: Beispiel für einen Harvester in fml40

6.3 Dashboard

Abbildung 6-3 zeigt das Beispiel einer Dashboard-MMS:

- mit der Rolle Dashboard (***ml40::Dashboard***)
- zunächst ohne weitere Spezifikation ihrer Features

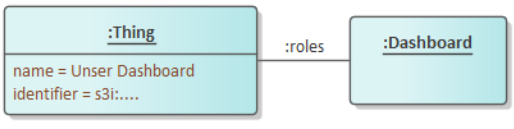

Abbildung 6-3: Beispiel für ein Dashboard in fml40

6.4 Fällhilfe-App

Abbildung 6-4 zeigt das Beispiel einer Fällhilfe-App, also eine WH4.0-MMS:

- mit der allgemeinen Rolle App (***ml40::App***)
- mit einer noch näher zu spezifizierenden Functionality zur Unterstützung von Fällmaßnahmen (***fml40::GeneratesFellingSuggestions***)

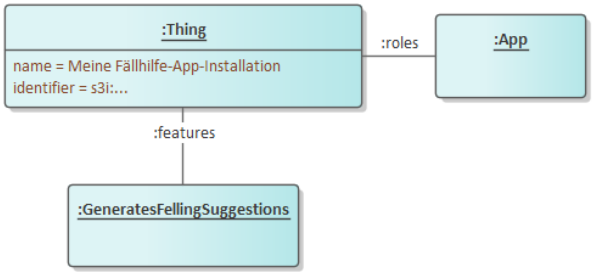

Abbildung 6-4: Beispiel für eine Fällhilfe-App in fml40

6.5 DZ einer Person

Abbildung 6-5 zeigt das Beispiel des DZ einer Person

- mit der Doppelrolle Waldbesitzer (***fml40::ForestOwner***) und Waldarbeiter (***fml40::ForestWorker***)
- mit ***Value***s für den aktuellen Ort (***ml40::Location***), den Namen der Person (***ml40::PersonalName***) sowie für zwei Adressen (***ml40::Address***) mit Bezugnahme auf die jeweilige Rolle (***relatesToRoles***)

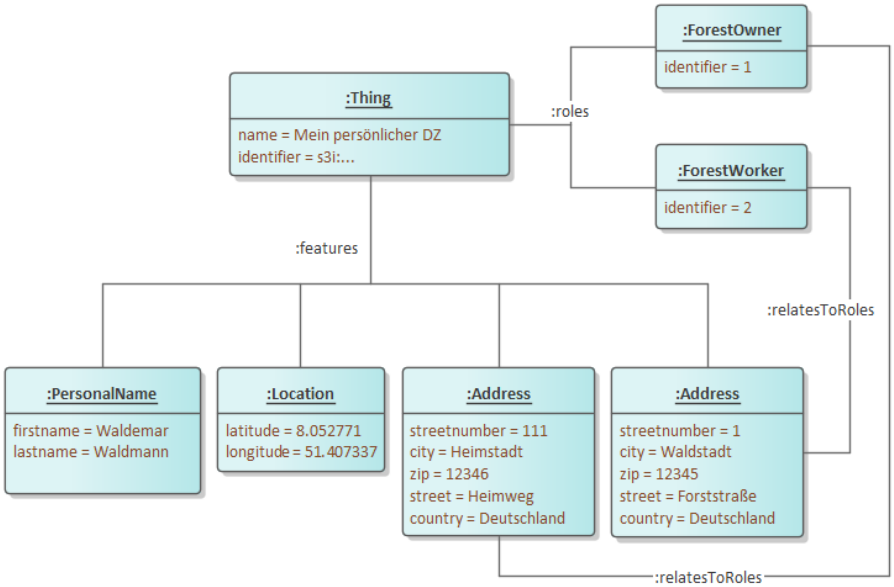

Abbildung 6-5: Beispiel für den DZ einer Person in fml40

6.6 DZ Wald

Abbildung 6-6 zeigt das Beispiel des DZ eines Waldes, genauer eines Waldsegments

- mit der Rolle (***fml40::ForestSegment***)
- mit einem Value für Inventurdaten (***fml40::InventoryData***); dieser wiederum referenziert Inventurdaten im ForestML-Format für Inventurdaten (***fml::AbstractInventory***)

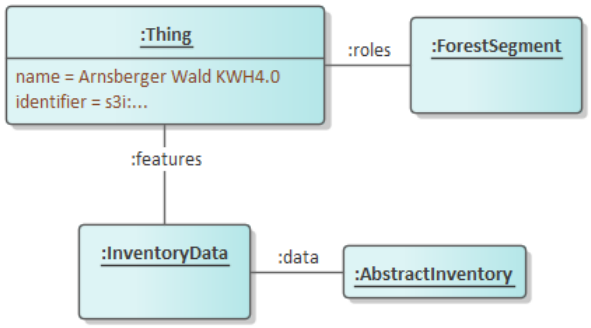

Abbildung 6-6: Beispiel für einen DZ Wald in fml40

6.7 Transportauftragssoftware

Abbildung 6-7 zeigt das Beispiel einer Transportauftragssoftware

- mit der allgemeinen Rolle App (***ml40::App***)
- mit einem generischen, herstellerspezifisch genutzten Value (***ml40::GenericProperty***) für die Version der genutzten Software
- mit einer Functionality für das Verteilen auf Aufträgen (***ml40::ManagesJobs***) – ausstehend ist die Spezifikation weiterer Functionalitys

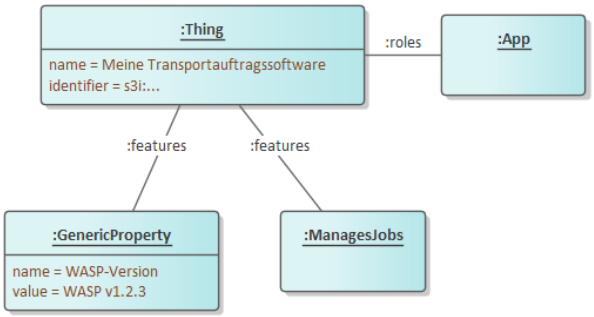

Abbildung 6-7: Beispiel für eine Transportauftragssoftware in fml40

7 Abbildung auf JSON

Die Abbildung des konzeptuellen Datenmodells auf JSON als physikalischem Datenmodell kann je nach Zielsystem unterschiedlich erfolgen. In diesem Abschnitt werden die zwei zentralen Abbildungen

1. auf Laufzeitumgebungen (S3I-Repository, Edge-basierten Lösungen …) sowie
2. auf das S3I-Directory (als „Telefonbuch“ der WH4.0-Dinge)

beschrieben. Dies soll am Beispiel des Harvesters aus Abschnitt 6.2 demonstriert werden.

7.1 Laufzeitumgebungen (S3I-Repository, Edge …)

In diesem Szenario soll das WH4.0-Ding selbst (*nicht* nur sein „Telefonbucheintrag“) in einer Laufzeitumgebung wie dem S3I-Repository, einer anderen Cloud- oder einer Edge-basierten Lösung auf JSON-Basis abgebildet werden. Dies bedingt eine weitestgehend ***direkte Abbildung*** des konzeptuellen Datenmodells von fml40 auf JSON (Abbildung 7-1).

Zu beachten ist dabei, dass diese technische Umsetzung des konzeptionellen Datenmodells in zwei Varianten genutzt werden kann:

- Zur Umsetzung einer ***Datenhaltung*** als deren internes Datenformat

- Zur ***Bereitstellung*** der in einem anderen internen Format verwalteten Daten über irgendeine Art von Schnittstelle ***nach außen***

Abbildung 7-1: Abbildung des Harvesters auf JSON z.B. für das S3I-Repository (Ausschnitt) – innerhalb des roten Kastens sind die ForestML 4.0-spezifischen Angaben, außerhalb die Ditto-konformen

Die Abbildung orientiert sich an den Grundstrukturen von Eclipse Ditto, auch wenn die Daten nicht zwingend mit Ditto verwaltet werden müssen. Der ***identifier*** des WH4.0-Dings wird auf die Ditto-***thingId*** abgebildet. Die ID-Werte selbst werden dabei vom S3I vergeben (daher der Präfix). Entsprechend wird eine Ditto-***policyId*** angegeben. Unter dem Ditto-spezifischen Schlüssel ***attributes*** folgt die komplette Beschreibung der Struktur des WH4.0-Dings (zusätzlich markiert durch den roten Kasten). Die Klassen von Objekten werden dabei über den Schlüssel ***class*** spezifiziert. Alle anderen Objekteigenschaften werden über gleichnamige Schlüssel abgebildet, z.B. ein Schlüssel ***longitude*** für das gleichnamige Attribut ***longitude*** des (f)ml40-Features ***ml40::Location***. Auch Beziehungen zwischen Objekten werden auf JSON-Schlüssel abgebildet, z.B. zeigt die Beziehung ***roles*** auf eine Liste von Objekten mit einem Eintrag (einem Rollenobjekt mit der Klasse ***fml40::Harvester***). Im Fall von ***ml40::Shared*** werden die Zielobjekte (***targets***) über die Angabe der Identifier (d.h. ***thingId***s) der Zielobjekts angegeben, da

es sich um eigenständige top-level WH4.0-Dinge handelt, die an dieser Stelle nicht untergeordnet verwaltet werden. Über ***ml40::Composite*** zugeordnete WH4.0-Dingen wie der Motor (und weitere) werden hingegen unmittelbar dem jeweils referenzierenden WH4.0-Ding untergeordnet. In diesem Fall wird der ***identifier*** auch direkt auf eine gleichnamige Eigenschaft abgebildet (es wird hier keine Ditto-***thingId*** genutzt).

7.2 S3I-Directory

Im vorstehenden Abschnitt werden die Daten eines WH4.0-Dings selbst verwaltet, im Beispiel also z.B. die Motordrehzahl.

Im S3I-Directory hingegen werden nur ***Metadaten*** des WH4.0-Dings verwaltet (Abbildung 7-2). Dazu wird ein eigenes Metamodell (das Datenmodell des S3I-Directory) verwendet, insbesondere um einzelnen Teilen des WH4.0-Dings eigene Endpoints zuordnen zu können, über die sie erreichbar bzw. abrufbar sind. Die ForestML 4.0-Strukturen werden also mithilfe dieses Metamodells auf JSON abgebildet (rot umrahmter Bereich), wodurch die Abbildung etwas „***indirekter***“ ist als im vorherigen Abschnitt. Weitere Details dazu finden sich im KWH4.0-Standpunkt zur Smart Systems Service Infrastructure (S3I)[3].

Wie in Abschnitt 6.1 bereits beschrieben sollen bei der Abbildung des konzeptuellen ForestML 4.0-Datenmodells auf das S3I-Directory lediglich die ***hierarchische Struktur*** eines WH4.0-Dings ***und seine Features*** (Values und Functionalitys) angegeben werden. Dadurch kann der Nutzer bei Einsicht in das S3I-Directory bereits erkennen, welche Features ein WH4.0-Ding aufweist. Die einzelnen Servicefunktionen der Functionalitys bzw. die Attribute der Values sollen nicht über S3I::Service- bzw. S3I::Value-Strukturen im S3I-Directory deklariert werden, weil sich diese bereits über die Spezifikation von fml40 ergeben.

[3] https://www.kwh40.de/veroffentlichungen/

```
{
    "thingId": "s3i:...",
    "policyId": "s3i:...",
    "attributes": {
        "ownedBy": "...",
        "name": "Mein Harvester",
        "location": {
            "longitude": 6.2492276,
            "latitude": 50.808797
        },
        "type": "component",
        "dataModel": "fml40",
        "allEndpoints": ["s3ib://s3i:..."],
        "thingStructure": {
            "class": "ml40::Thing",
            "links": [
                {
                    "association": "roles",
                    "target": {"class": "fml40::Harvester"}
                },
                {
                    "association": "features",
                    "target": {"class": "fml40::AcceptsProximityAlert"}
                },
                {  ...
                {
                    "association": "features",
                    "target": {
                        "class": "ml40::Composite",
                        "links": [
                            {
                                "association": "targets",
                                "target": {
                                    "class": "ml40::Thing",
                                    "identifier": "s3i:...",
                                    "links": [
                                        {
                                            "association": "roles",
                                            "target": {"class": "ml40::Engine"}
                                        },
                                        {
                                            "association": "features",
                                            "target": {"class": "ml40::RotationalSpeed"}
                                        }
                                    ]
                                }
                            },
                            { [35 lines]
                            { [10 lines]
                        ]
                    }
                }
            ]
        }
    }
}
```


Abbildung 7-2: Abbildung des Harvesters auf JSON für das S3I-Directory (Ausschnitt)