FORCAM FORCE™
Bridge API

Version 5.11

Handbuch

Dolument: Handbuch - Bridge API.docx

Freigabedatum: 2021-02-10

Dokumentversion: V1

Autor: STernes

FB_8-41 ERSTELLT AM: 10.02.21 GEÄNDERT AM: 24.02.21 VERSION: V1
FREIGEGEBEN DURCH: HEAD OF TTE AUTOR: TECHNICAL WRITER VERTRAULICHKEITSSTUFE: ÖFFENTLICH

Inhalt

1 Einführung

FORCE Bridge API ist die Programmierschnittstelle für eine Produktionsumgebung im so genannten Industrial Internet of Things (IIoT). Als solches dient sie als Schnittstelle zwischen den Objekten der Produktionsumgebung und der IT-Infrastruktur eines Produktionswerks.
FORCE Bridge API dient als Brücke zwischen dem realen Produktionsstandort, den IT-Systemen und den Anwendungen, die für eine effektive und effiziente Organisation eingesetzt werden.

Die Industrie IoT-Plattform erfüllt zwei Funktionen:

- Die FORCE Bridge API liefert ein vollständiges digitales Abbild einer Produktionsanlage mit ihren relevanten Objekten samt ihren Zuständen. Objekte sind reale Einheiten wie Personen, Maschinen und Werkzeuge, aber auch abstrakte Einheiten wie Produktionsaufträge oder Vorgänge.
- Die FORCE Bridge API gewährleistet die organisatorische Interoperabilität[1] von Personen, Produktionsanlagen und IT-Systemen einer Produktionsanlage zur Maximierung der Ressourceneffektivität und Prozesseffizienz.

Diese Dokumentation der API besteht aus verschiedenen in sich geschlossenen, aber aufeinander aufbauenden Teilen. Jeder dieser Teile befasst sich mit einem bestimmten Abschnitt der API, erklärt zunächst den technischen Hintergrund und endet mit einem oder mehreren Beispielprogrammen unter Verwendung des Application Development Kit (ADK).

1.1 Technisches Grundwissen

FORCE Bridge API verbindet Ihre Anwendung mit einer IoT-Plattform. Ihre Anwendung kann die API verwenden, um auf Daten zuzugreifen, Befehle an die FORCE Runtime zu senden oder um auf Ereignisse zu warten, die im System auftreten.

- Der komplette Zugriff auf die API erfolgt über HTTP(S).
  Anforderungs-Payloads werden als *application/json* formatiert.
  Antwort-Payloads sind als *application/json* oder *application/hal+json* formatiert.
- **OAuth2** wird für alle Authentifizierungen verwendet.
  Alle API-Anforderungen müssen authentifiziert werden, ansonsten gibt es eine *401 Unauthorized* Fehlerantwort (siehe **Authentication**).
- FORCE Bridge API ist eine **erkundbare API** und stellt alles als Ressource bereit.
- Jede Ressource wird durch ihren universellen eindeutigen Bezeichner (**UUID = universal unique identifier**) identifiziert.

ⓘ Die komplette Swagger-Spezifikation der FORCE Bridge API ist hier verfügbar.

[1] Interoperabilität beschreibt die Fähigkeit, Informationen zwischen zwei unterschiedlichen Systemen auszutauschen und nutzbar zu machen.

1.1.1 Erkundbare API

Antwort-Payloads werden mit Hyperlinks unter Verwendung der **Hypermedia Application Language** (HAL) versehen. HAL ist ein einfaches Format, das eine Reihe von Konventionen zum Ausdrücken von Hyperlinks in JSON bereitstellt - im Grunde ist es JSON mit Hyperlinks. Dies ermutigt Entwickler, Anwendungen zu erstellen, die diese Hyperlinks anstatt hart kodierte URLs verwenden, um zwischen Ressourcen zu navigieren.

Die folgenden *Felder* sind bei jeder Ressource zu erwarten:

- *properties* – Die regulären Eigenschaften der Ressource
- *_links* – Ein JSON-Objekt mit einem Satz von URL-Pfaden relativ zur API-Basis-URL
- *_embedded* – Ein JSON-Objekt mit anderen Ressourcen, die in die aktuelle Ressource eingebettet sind.

1.1.1.1 Kollektionen

Ein Satz von Ressourcen ist eine *Kollektion*.

1.1.2 Revisionierung

Die API wird durch Bezeichner im URI überarbeitet.
Die Version der API wird nur dann revidiert (erhöht), wenn eine wesentliche Änderung in einem der Webservices auftritt. Wesentliche Änderungen sind:

- Entfernung einer JSON Eigenschaft
- Neubenennung einer JSON Eigenschaft
- Allgemeine Umstrukurierung des Designs bestehender Darstellungen

Wesentliche Änderungen sind nicht:

- JSON Eigenschaften hinzufügen
- Neue Webservices hinzufügen

ⓘ Die Revisionierung ist global. Das bedeutet, dass ein Webservice dieselbe Antwort für verschiedene Versionen zurückgeben kann, wenn an diesem spezifischen Webservice keine Änderung vorgenommen wurde.

Möglicherweise fügt eine neue Version neue JSON-Eigenschaften zur Antwort hinzu. Dadurch wird die Webservice-Version nicht erhöht. Um die Abwärtskompatibilität zu gewährleisten, sollte jeder Client, der mit der Bridge-API kommuniziert, nur bei Bedarf neue Eigenschaften verarbeiten.

1.1.3 HTTP Anfragen

Nach Möglichkeit verwendet die API für jede Aktion geeignete **HTTP Methoden.**

Methode	Beschreibung
GET	Für das Abrufen von Ressourcen.
POST	Zum Erstellen von Ressourcen oder Ausführen einer Aktion, u.a.
PUT	Für das Ersetzen von Ressourcen.
DELETE	Für das Löschen von Ressourcen.

Überschriften

Schlüssel	Wert
authorization	Base64-kodierte Client-ID und Client-Geheimnis
accept header	*application/json* oder *application/hal+json*
accept language	*en-GB* (Standard), *en-US, de-DE, zh-CN*

1.1.4 HTTP-Statuscodes

Die API ist bestrebt, für jede Anfrage entsprechende **HTTP-Statuscodes** zurückzugeben.

Code	Beschreibung
200 OK	Erfolg
201 Created	Die Anfrage war erfolgreich und es wurde eine neue Ressource erstellt. Die Standort-Überschrift der Antwort enthält den URI der neuen Ressource.
202 Accepted	Die Anfrage wurde akzeptiert und ist in Bearbeitung, aber noch nicht abgeschlossen
204 No content	Die Anfrage war erfolgreich, aber der Server gibt keinen Inhalt zurück. Für die meisten DELETE-Anfragen ist dies die häufigste Antwort.
400 Bad request	Die Anfrage war ungültig oder kann nicht bearbeitet werden. Eine begleitende Fehlermeldung erklärt mehr.
401 Unauthorized	Die Authentifizierungsberechtigung fehlte oder war falsch.
403 Forbidden	Die Anfrage wird erkannt, aber sie wurde abgelehnt oder der Zugriff ist nicht erlaubt. Eine begleitende Fehlermeldung erklärt mehr.
404 Not found	Die angeforderte URI ist ungültig oder die angeforderte Ressource (z. B. ein Ereignis) existiert nicht.

Code	Beschreibung
415 Unsupported media type	The request entity has a content type that the server does not support.
500 Internal server error	Etwas ist fehlerhaft.
504 Gateway timeout	Der als Gateway oder Proxy fungierende Server hat keine rechtzeitige Antwort von dem Upstream-Server erhalten, auf den er zugreifen musste, um die Anfrage zu vervollständigen.

1.1.5 Authentifizierung

Die FORCE Bridge API verwendet das OAuth 2.0-Protokoll zur Authentifizierung.

⚠ OAuth ist nur sicher, wenn es über das TLS/SSL-Protokoll verwendet wird, daher müssen alle Anfragen und API-Endpunkte HTTPS verwenden..

⚠ Token sind Passwörter: Die Client-ID, das Client-Geheimnis und die Zugriffstoken Ihrer Anwendung sollten als genauso sensibel wie Passwörter betrachtet und nicht weitergegeben werden.

Beim Zugriff auf die FORCE Bridge API, folgen alle Anwendungen einem Grundmuster.
Im allgemeinem werden diese drei Schritte befolgt:

- Zugriffstoken beziehen
  Um eine Authentifizierungsanfrage an die API zu stellen, muss Ihre Anwendung zunächst unter Mithilfe des Client Credentials Flow, ein OAuth Zugriffstoken erhalten.
- Zugriffstoken für die Ausführung von Anfragen verwenden
  Nachdem Ihre Anwendung ein Zugriffstoken erhalten hat und sie API-Anfragen stellt, sendet sie die *access_token* -Eigenschaft in einer Token-Antwort in einem **Bearer authorization header** (DE: Träger-Authorisierungs-Überschrift).
- Zugriffstoken beim Ablauf aktualisieren
  Zugriffstoken haben eine begrenzte Lebensdauer. Wenn Ihre Anwendung nach Ablauf der Lebensdauer eines Zugriffstokens eine Anfrage stellen muss, muss sie ein neues Zugriffstoken erhalten.

1.2 FORCAM FORCE™ Bridge API verwenden

Im Gegensatz zu allen anderen HTTP-Methoden muss POST für verschiedene Zwecke verwendet werden, um in bestimmten Situationen unbegrenzte Aufrufe zu vermeiden.

Im engeren Sinne wird POST verwendet, um eine oder mehrere neue Ressourcen zu erzeugen. Dazu wird die entsprechende Sammlung mit POST aufgerufen:
POST/{collection}

Nach dem Hypermedia-Prinzip der RESTful-Architektur wird bestimmt, ob eine Ressource erstellt werden kann oder nicht, wenn die zugehörige Sammlung einen entsprechenden Hyperlink hat. Zum

Beispiel bietet die Bridge-API in der Sammlung der Werkzeug-Montageaufträge einen Hyperlink zur folgenden Methode, mit der Anwendungen neue Werkzeuge erstellen können:
POST/tools/assemblyOrders

Alle Eigenschaften des Werkzeugmontageauftrags werden mit dem Methodenaufruf an die IoT-Plattform übertragen, die wiederum mit einer Darstellung der neu angelegten Ressource antwortet, wenn der Aufruf erfolgreich war. Insbesondere wird in der Antwort die UUID des neu erstellten Werkzeugs bekannt gegeben. Eine weitere Verwendung von POST ist das Ändern des Ressourcenzustands. Dazu wird die betreffende Einzelressource mit POST und einer Angabe des neuen Zustands aufgerufen:
POST/{collection}/{id}/{state}

Die Bridge-API ermöglicht es einem Werkzeugverwaltungssystem, den Zustand eines Werkzeugmontageauftrags z. B. durch den folgenden Aufruf zu setzen:
POST/tools/assemblyOrders/{toolAssemblyOrderId}/{toolAssemblyOrderStateId}

Obwohl auch die PUT- oder PATCH-Methode für diesen Zweck verwendet werden könnte, wären die Aufrufe wesentlich umständlicher, da bei PUT auch Eigenschaften übergeben werden müssen, die nicht geändert werden. Andererseits ist es schwierig, dies mit dem Hypermedia-Konzept einer **REST**-Architektur zu vereinbaren. Letzteres verlangt, dass einer Anwendung, die eine Ressource repräsentiert, alle möglichen Zustandsübergänge als Hyperlinks präsentiert werden und dass jeder Zustandsübergang durch das Folgen eines Hyperlinks ausgelöst wird. Es ist möglich, den folgenden Verweis in HAL zu erstellen:

```
1.  "updateToolAssemblyOrderState": {
2.     "method": "POST",
3.     "embeddable": false,
4.     "href": ".../tools/assemblyOrders/E446BB6B0C084CB09D0BC0319A8A1F1C/AVAILABLE"
5.  }
```

Alternativ dazu müsste bei einer PUT- oder PATCH-Methode die Angabe des neuen Zustands im Anfrageinhalt angegeben werden.

Wenn hunderte von Ressourcen in einer einzigen Transaktion aktualisiert werden sollen, würde der wiederholte Aufruf der PUT-Methode unnötigen Kommunikationsaufwand und damit Verzögerungen bedeuten, da mit PUT immer nur eine Ressource angesprochen werden kann.

Außerdem besteht die Gefahr, dass sich die Daten in der IoT-Plattform in einem inkonsistenten Zustand befinden, bis die Aufrufsequenz abgeschlossen ist. Um dies zu vermeiden, wird für bestimmte Zwecke auch eine transaktionssichere POST-Methode als Alternative zum wiederholten Aufruf der PUT-Methode angeboten. In diesem Fall wird die folgende Notation verwendet, um anzuzeigen, dass die Ressourcen verarbeitet werden:
POST/operations/forecastResults/update

Es gibt noch zwei weitere Beispiele in der Bridge API:
POST/operations/planningResults/update
POST/staffMembers/planningResults/update

Zuletzt wird auch POST anstelle der GET-Methode verwendet. Der Grund dafür ist, dass die GET-Methode nicht geeignet ist, eine große Anzahl von Filterparametern beim Aufrufen zu setzen. Dieses Problem tritt auf, wenn die Sammlungen von Fertigungsaufträgen oder Vorgängen explizit nach hunderten von Auftragsnummern gefiltert werden sollen. Für den Abruf beider Kollektionen ist daher

auch die POST-Methode vorgesehen. Die folgende Notation wird verwendet, um zu zeigen, dass eine solche POST-Methode für den Abruf einer Sammlung und nicht für die Erstellung einer neuen Ressource vorgesehen ist:
POST{collection}/search

1.2.1 OAuth-Token anfordern

Um ein Zugriffstoken zu erhalten, das den Zugriff auf die FORCE Bridge API ermöglicht, müssen Anwendungen den **Client Credentials Flow der OAuth2-Spezifikation** verwenden.

- Einfache Autorisierungs-Überschrift erstellen:
  URL-kodieren Sie die Client-ID und das Client-Geheimnis Ihrer Anwendung gemäß **RFC 1738**.
  Verketten Sie die verschlüsselte Client-ID, ein Doppelpunkt-Zeichen (:) und das verschlüsselte Kundengeheimnis in eine einzige Zeichenfolge. Fügen Sie "Basic" an die Zeichenfolge an.
  Kodieren Sie die Zeichenfolge aus dem vorherigen Schritt in Base64

Beispiel

Überschrift:

{ Authorisation : Basic xyz Inhalt_Typ: Applikation/x-www-form-urlencode } Body: ‘grant_type=client_credentials&scope =read write’

Beispiel-Code

```
{
        "// How to create Authorization String"
        "var encodedClientID = encodeURI(clientID); //encode with URLEncode"
        "var encodedClientSecret = encodeURI(clientSecret); //encode with URLEncode"
        "var pair = encodedClientID+':'+encodedClientSecret; //combine Strings with
    Semicolon"
        "var encodedPair = window.btoa(pair); //encode with Base64"
        "var authorizationValue = 'Basic' +encodedPair; //add Basic to the String"

}
```

- Zugriffstoken erhalten
  Der verschlüsselte Wert muss gegen ein Zugriffstoken ausgetauscht werden.

1.2.2 POST-Anfragen versenden

Die Token URL ist:

POST http(s)://$HOST:$PORT/ffwebservices/oauth/token

Überschriften

Name	Wert
Authorization	“Basic” + Base64 verschlüsselte clientid:clientsecret vom Vorschritt
Content type	*application/x-www-urlencode*

Parameters

Name	Typ	Wert
grant_type	*string*	Erforderlich. Wert sollte *client_credentials* sein
scope	*string*	Leerzeichenbegrenzte Zeichenfolge der gewünschten Bereiche

Scopes

Name	Beschreibung
read	Erteilt die Berechtigung zum Aufruf von HTTP GET
write	Erteilt die Berechtigung zum Aufruf von HTTP POST, PUT, DELETE

⚠ Ablauf des Tokens: Anwendungen sollten unter Berücksichtigung der Möglichkeit geschrieben werden, dass ein gewährter Token möglicherweise nicht mehr funktioniert. Die Anwendung muss sich erneut authentifizieren, um neue Zugriffstoken zu erhalten. Die Zeit, bis ein Token abläuft, wird mit der Token-Antwort zurückgegeben.

Java

ⓘ Diese Aufgabe wird automatisch ausgeführt, wenn eine Anfrage gestellt wird, wobei die bei der Initialisierung des *BridgeAPI* -Objekts angegebenen Anmeldeinformationen verwendet werden.

CURL

```
curl -X POST http(s)://$HOST:$PORT/ffwebservices/oauth/token \
     --header "accept:application/x-www-urlencode" \
     -d "client_id=$CLIENT_ID" \
     -d "client_secret=$CLIENT_SECRED" \
     -d "grant_type=client_credentials" \
     -d "scope=read%20write"
```

Beispiel-Anfrage

```
{
        "//access_token must be requested first"
        "method": "POST",
        "headers": "{",
        "content-type": "application/json",
        "Authorization": "'Bearer ' +access_token"
}
```

Beispiel-Antwort

```
{
        "access_token": "537517ab-faa3-4ad2-8ae5-37ff91ffb7c0",
        "token_type":   "bearer",
        "expires_in":   42523,
        "scope":        "read write"
}
```

Fußnoten:
[1] Interoperabilität beschreibt die Fähigkeit, Informationen zwischen zwei unterschiedlichen Systemen auszutauschen und nutzbar zu machen.

2 Bridge API als digitales Abbild einer Produktionsstätte

In diesem Abschnitt werden die Konzepte und Objekte einer Produktionsstätte beschrieben. Es wird erklärt, wie sie in der FORCAM FORCE™ Bridge API dargestellt werden und wie auf sie zugegriffen und mit ihnen gearbeitet wird.

2.1 Grundkonzepte und Objekte in der Produktion

Ziel des IoT ist es, ein digitales Abbild der realen Welt und ihrer Objekte zu schaffen. Diese Objekte fungieren einerseits als konkrete materielle Gegenstände und andererseits als abstrakte Konzepte. Ein IoT für den Bereich der Produktion erzeugt ein digitales Abbild sowohl der Maschinen und Werkzeuge, die in der Produktionsstätte eingesetzt werden, als auch der Konzepte, die den Produktionsalltag dominieren, wie z. B. Produktionsaufträge und Arbeitsvorgänge.

2.1.1 Einheiten aus der Produktion

2.1.1.1 Fertigungsaufträge

Wenn ein bestimmtes Produkt an einem bestimmten Punkt mit einer bestimmten Menge produziert werden soll, gibt das Produktionsplanungssystem die Erstellung eines Fertigungsauftrags aus. Die Vorlage für den Fertigungsauftrag ist ein Arbeitsplan, der jeden Schritt (Vorgang) enthält, der zur Herstellung des Endprodukts erforderlich ist. Der Arbeitsplan legt fest, an welcher Art von Arbeitsplatz oder Maschinen der Vorgang ausgeführt werden soll, sowie wie viel Zeit und welche Komponenten benötigt werden.

Die Erstellung eines Fertigungsauftrags läuft folgendermaßen ab:

- Das Unternehmen erhält einen oder mehrere Kundenaufträge zur Herstellung eines bestimmten Produkts oder möchte eine bestimmte Menge davon auf Lager produzieren.
- Im **ERP**-System wird ein neuer Fertigungsauftrag mit einer eigenen Nummer angelegt, der alle Informationen aus dem Arbeitsplan enthält.
- Der Fertigungsauftrag enthält zusätzlich zu den Angaben im Arbeitsplan die Angabe der Menge des zu produzierenden Produkts, diese wird als Sollmenge bezeichnet.
- Schließlich wird aus dem Liefertermin ein Eckendtermin für den Fertigungsauftrag erzeugt und das Ergebnis der Materialbedarfsplanung ist ein Eckstarttermin. Der Starttermin entspricht dem Zeitpunkt, zu dem alle für die Bearbeitung des Fertigungsauftrags benötigten Komponenten verfügbar sind.

2.1.1.2 Arbeitsvorgänge

Ein Arbeitsvorgang beschreibt einen einzelnen Schritt, der zur Bearbeitung eines Fertigungsauftrags erforderlich ist. Jeder Arbeitsvorgang produziert ein bestimmtes Material in einer vorgegebenen Menge (Zielmenge des Arbeitsvorgangs). Dabei handelt es sich entweder um ein Material, das in einem nachfolgenden Vorgang als Komponente weiterverarbeitet wird, oder im Falle eines letzten Arbeitsgangs um das Endprodukt des Fertigungsauftrags.

Zu den im Arbeitsgang gespeicherten Standardwerten gehören:

- Das zu fertigende *Material*.
- Die *Sollmenge* ist die Menge des zu produzierenden Materials. Diese kann sich von der Sollmenge des Fertigungsauftrags unterscheiden (z. B. 4 Türen, die in einem Arbeitsgang gefertigt werden müssen, um letztendlich ein Automobil zu produzieren).
- Die *Mengeneinheit* der Sollmenge (z. B. Stück, Meter, Kilogramm, etc.).
- Der *Arbeitsplatz* oder eine *Gruppe* von gleichwertigen Arbeitsplätzen, an denen der Vorgang bearbeitet werden soll.
- Die *Sollzeit pro Einheit*, d.h. die Zeit, die für die Herstellung einer Einheit des zu produzierenden Materials benötigt wird.
- Die *Sollbearbeitungszeit* ist die Zeit, die für die gesamte Bearbeitungszeit des Vorgangs verwendet wird. Sie ist das Produkt aus Sollmenge und Sollzeit pro Einheit.
- Die *Soll-Rüstzeit* ist die Zeit, die für das Rüsten des Arbeitsplatzes eingeplant ist.
- Die *Soll-Abrüstzeit* ist die Zeit, die für den Abbau des Arbeitsplatzes eingeplant ist.
- Die *Soll-Transportzeit* ist die Zeit, die für den Transport zum nächsten Arbeitsplatz eingeplant ist.
- Die *Soll-Liegezeit* ist die Zeit, die für Funktionen wie Kühlen, Trocknen oder Aushärten des zu produzierenden Materials eingeplant ist.
- Die *Komponenten*, die benötigt werden, um das Material zu produzieren.
- Die *Fertigungsresourcen- und Hilfsmittel* (z. B. Werkzeuge, Geräte oder Dokumente), die am Arbeitsplatz benötigt werden, um den Arbeitsvorgang auszuführen.

2.1.1.2.1 Vorgangsphasen

Die Vorgangsphasen beschreiben den Zustand eines Vorgangs; von der Freigabe durch das ERP-System bis zur Fertigstellung und Rückbuchung der Ausgangsmengen des Vorgangs an das ERP-System.

2.1.1.3 Arbeitsplätze

Als Minimum wird jeder Arbeitsvorgang an einem Arbeitsplatz ausgeführt. Ein Arbeitsplatz kann ein einfacher Handarbeitsplatz oder ein Arbeitsplatz mit einer automatisierten Fertigungslinie (Maschine) sein.

2.1.1.4 Arbietsplatz Gruppen

Arbeitsplätze werden in ERP-Systemen oft als ein einzelner Arbeitsplatz mit mehreren Kapazitäten hinterlegt. In Bridge API wird jeder Arbeitsplatz einzeln angezeigt und dann einer bestimmten Kapazitätsgruppe bzw. Produktionslinie zugeordnet.

- Kapazitätsgruppen sind Arbeitsplätze, an denen das gleiche Produktionsverfahren ausgeführt wird, und daher zusammen gruppiert werden. Zum Beispiel gehört ein Spritzgießarbeitsplatz zur Kapazitätsgruppe Spritzgießmaschinen.
- Produktionslinien sind verschiedene Arbeitsplätze, die kollektiv einen kompletten Produktionsprozess eines Produkts abbilden. Hierbei werden Prozesse der Linienfertigung oder Massenfertigung verwendet. Dabei sind die verschiedenen Stationen der Produktionslinie nicht durch zeitliche Taktung des Produktionszyklus verbunden, sondern Pufferspeicher fangen die Zeitunterschiede bei der Bearbeitung zwischen benachbarten Stationen auf. So kann z. B. ein fehlerhaftes Produkt zur Nacharbeit aus dem Produktionsablauf herausgenommen werden, ohne den Produktionsprozess zu stören.

Jeder Arbeitsplatz kann genau einer Kapazitätsgruppe und maximal einer Fertigungslinie zugeordnet werden.

2.1.1.5 Mitarbeiter

Die Mitarbeiter sind das Produktionspersonal, das in Schichten eingeteilt ist und den jeweiligen Arbeitsplätzen zugeordnet werden kann. Die Mitarbeiter tragen neben ihren Personaldaten zusätzliche Informationen wie z.B. Qualifikationen, gebuchter Urlaub, etc. mit sich, die für die Planung und den Ablauf der Produktion wichtig sind.

2.1.1.6 Werkzeuge

Produktionswerkzeuge werden an Maschinenarbeitsplätzen benötigt, um einen Vorgang fachgerecht durchzuführen. Ein Werkzeug ist z. B. die Spritzgussform einer Spritzgießanlage. Das Werkzeug hat einen Lagerort, einen aktuellen Einsatzort und eine Lebensdauer. Die Lebensdauer beschreibt, wie viel Zeit verbleibt, bis das Werkzeug ausgetauscht werden muss.

2.1.1.7 Dokumente

Für die Ausführung eines Vorgangs werden Dokumente benötigt, die in Produktionsordnern abgelegt werden. Diese Dokumente stammen in der Regel aus dem Product-Lifecycle-System. Bei den Dokumenten kann es sich um NC-Programme zur Steuerung von Werkzeugmaschinen handeln, aber auch um andere Dokumente wie Prüfpläne, Montageanleitungen oder Montagezeichnungen.
Werden Dokumente geändert, legt die IoT-Plattform automatisch eine neue Version an.
Alle Dokumente, die zu einem Vorgang gehören, werden im Shop Floor Terminal angezeigt. Von dort können auch die NC-Programme aus dem Fertigungsordner in die Maschinensteuerung geladen werden.

2.1.1.8 Geräte

Geräte sind alle mit der IoT-Plattform verbundene Geräte. Dazu gehören insbesondere die speicherprogrammierbaren Steuerungen der angeschlossenen Maschinen und Produktionsanlagen. Diese sind fest einem bestimmten Arbeitsplatz zugeordnet.

Mit **GET/workplaces/{workplaceId}/devices** können alle Geräte abgerufen werden, die an einem Arbeitsplatz oder einer Maschine vorhanden sind. Zu den Geräten können alle aufgezeichneten Prozess- und Sensordaten abgefragt werden.

2.1.2 Digitales Abbild in der FORCE Bridge API

Bridge API bildet das komplette digitale Abbild einer Produktionsanlage ab. Der RESTful [2]-Webservice kann direkt über einen Webbrowser aufgerufen werden, um mit den in der API bereitgestellten Ressourcen zu kommunizieren.

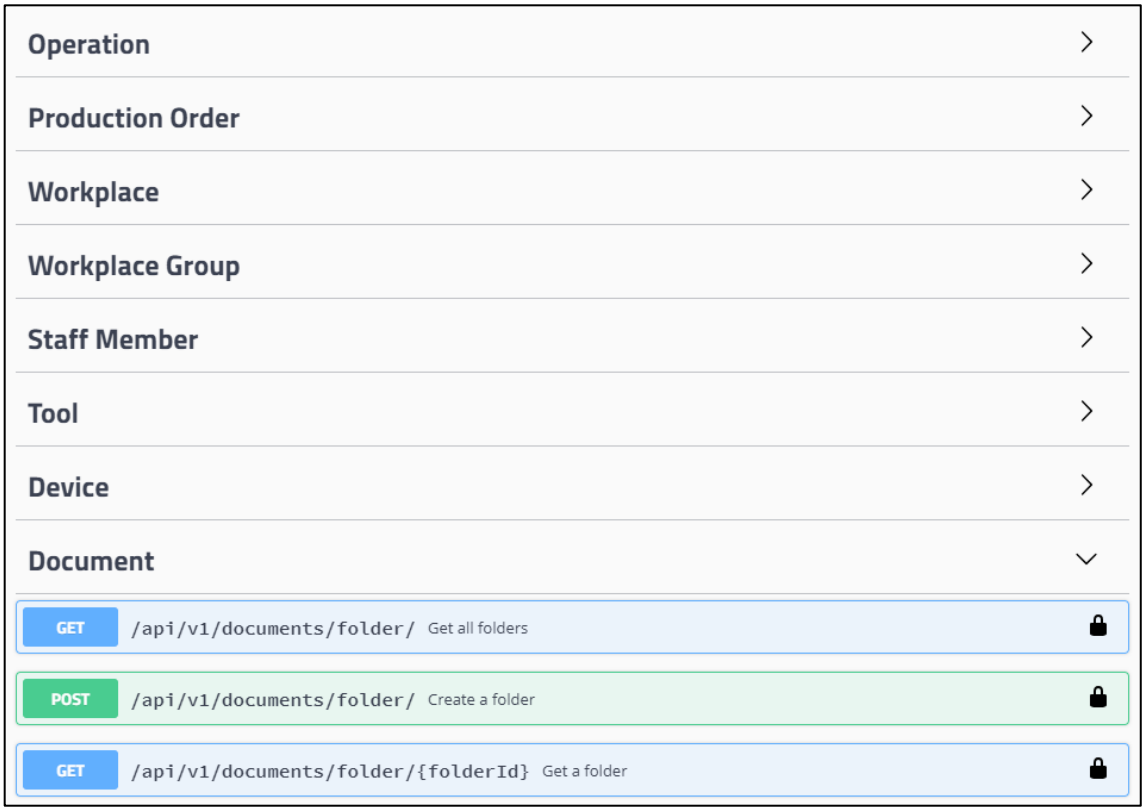

Bild 1: Zugriff auf API-Ressourcen über die Swagger UI

2.1.2.1 Der Einstiegspunkt und die Top-Level-Ressourcen

RESTful hat generell nur einen *Einstiegspunkt* (API-Basis), von woher die anderen ausgehenden Ressourcen abgerufen werden können. Um die API-Ressourcen finden zu können, ist es daher wichtig, dass der Uniform Resource Indicator (URI) mit den REST-Clients korrekt kommuniziert wird.

[2] RESTful sind Anwendungen oder Webservices, die Zustände und Funktionalitäten als eine Sammlung von Ressourcen anbieten und so die Interoperabilität zwischen Computersystemen im Internet ermöglichen.

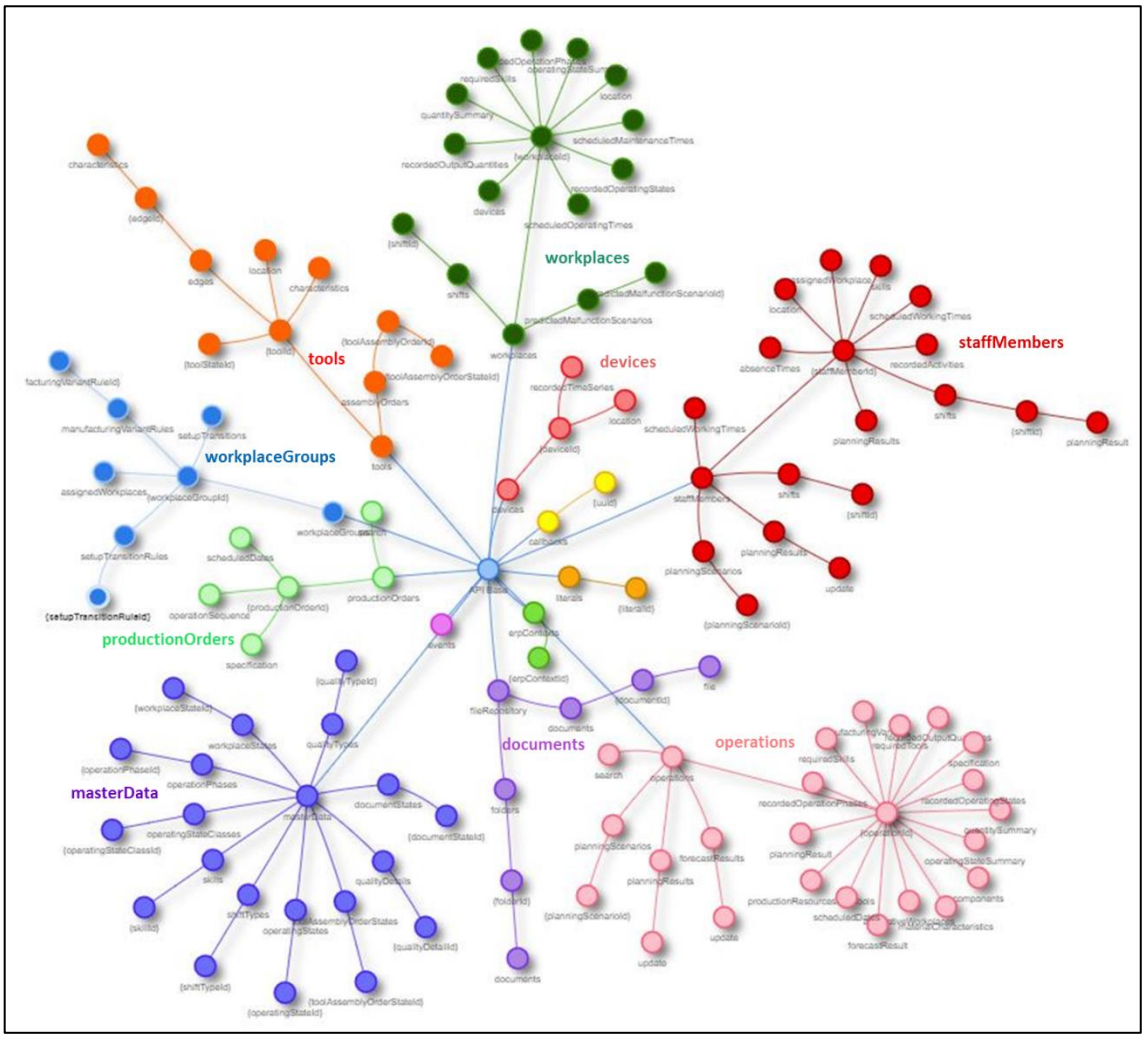

Bild 2: Grafische Darstellung der FORCE Bridge API Version 2.0

In diesem Abschnitt werden die folgenden Top-Level-Ressourcen näher beschrieben:

- *productionOrders/* Produktionsaufträge aus dem ERP-System.
- *operations/* Vorgänge der Fertigungsaufträge.
- *workplaces/* Arbeitsplätze (Maschinen oder Handarbeitsplätze), an denen Vorgänge ausgeführt werden.
- *workplaceGroups/* Kapazitätsgruppen (funktional ähnliche Maschinen) oder Produktionslinien.
- *staffMembers/* Die Mitarbeiter, die für die Ausführung von Operationen bereitstehen.
- *tools/* Die verfügbaren Werkzeuge für die Ausführung von Operationen.
- *documents/* Für die gesamte Produktionsanlage wichtige Dokumente.
- *devices/* Geräte wie z. B. speicherprogrammierbare Steuerungen von Maschinen.

2.1.3 Verwendung des Application Development Kit

2.1.3.1 Einstiegspunkt abrufen

Java

Um zu kommunizieren, muss zuerst eine *BridgeAPI* -Klasse mit der URL des Webdienstes und seinen Anmeldeinformationen eingerichtet werden.

```
1.      String url = "http://localhost:24080";
2.      String user = "myUser";
3.      String secret = "mySecret";
4.
5.      BridgeAPI api = new BridgeAPI(url, user, secret);
6.
7.      /* ... */
```

Danach werden die einzelnen Clients (aus Abschnitt 2.1.2.1) aus dem neu eingerichteten Objekt geholt.

```
1.  IProductionOrderClient productionOrderClient = api.getProductionOrderClient();
2.  IOperationClient operationClient = api.getOperationClient();
3.  IWorkplaceClient workplaceClient = api.getWorkplaceClient();
4.  IWorkplaceGroupClient workplaceGroupClient = api.getWorkplaceGroupClient();
5.
6.  IDeviceClient deviceClient = api.getDeviceClient();
7.  IToolClient toolClient = api.getToolClient();
8.  IMasterDataClient masterDataClient = api.getMasterDataClient();
9.
10. IStaffMemberClient staffMemberClient = api.getStaffMemberClient();
11.
12. IOperationPlanningClient planningScenarioClient = api.getPlanningScenarioClient();
13.
14. IEventClient eventClient = api.getEventClient();
15. ICallbackClient callbackClient = api.getCallbackClient();
16.
17. ICommandClient commandClient = api.getCommandClient();
18. ICommandClientAsync commandClientAsync = api.getCommandClientAsync();
19.
20. IMachineClient machineClient = api.getMachineClient();
21. IMachineStateDetailClient machineStateDetailClient =
    api.getMachineStateDetailClient();
22. IStatusDefinitionClient statusDefinitionClient = api.getStatusDefinitionClient();
23. IMiscellaneousClient miscClient = api.getMiscellaneousClient();
```

ⓘ Die Autorisierung erfolgt automatisch, wenn eine Anfrage gesendet wird.

CURL

```
1.  curl -X POST http(s)://$HOST:$PORT/ffwebservices/oauth/token \
2.       --header "accept:application/x-www-urlencode" \
3.       -d "client_id=$CLIENT_ID" \
4.       -d "client_secret=$CLIENT_SECRED" \
5.       -d "grant_type=client_credentials" \
6.       -d "scope=read%20write"
```

⚠ Ein gültiger Autorisierungs-Token ist für jeden Ressourcenzugriff erforderlich.

2.1.3.2 Ressourcen abrufen

Jede Ressource wird wie folgt abgerufen:

Java

```
1.  final GetProductionOrdersRequest ordersRequest = new GetProductionOrdersRequest();
2.  final Page<ProductionOrderResponse> orderResponsePage =
    productionOrderClient.getProductionOrders(ordersRequest);
3.
4.  /* Retrieve the elements of the first page */
5.  final List<ProductionOrderResponse> orderResponses = orderResponsePage.getElements();
6.  for (ProductionOrderResponse orderResponse : orderResponses) {
7.  final OrderPropertiesWSModel orderProperties = orderResponse.getProperties();
8.
9.  final String id = orderProperties.getId();
10. final String description = orderProperties.getDescription();
11. /* ... */
12. }
```

CURL

```
curl -X POST "http(s)://$HOST:$PORT/ffwebservices/api/v2/$resource"
  -H "Authorization: Bearer $TOKEN"
  -H "Accept-Language: en-US"
```

2.1.3.3 Seitennummerierung

Jede Sammlung von Ressourcen kann mehrere tausend Elemente enthalten. Daher wird das Konzept der Seitennummerierung angewandt, so werden Elemente in mehreren kleineren Sammlungen aufgeteilt, um übergroße Antworten zu vermeiden. Jede Antwort wird in mehreren überschaubaren Seiten aufgeteilt, mit maximal einhundert Elementen pro Seite. Mit dem Parameter *Limit* können benutzerdefinierte Seitengrößen für die API-Antworten festgelegt werden und mit dem Parameter *Offset* können die Daten durchgeblättert werden. API-Antworten geben vorgefertigte Seitenumbruch-Links mit Beziehungen zur *ersten*, *nächsten*, *vorherigen*, *letzten* und *aktuellen* Seite zurück, wobei die Client-Anwendungen aufgefordert werden, diesen Links für den Seitenumbruch zu folgen.

Java

Mit dem Java ADK gibt jede Sammel-Ressource ein *Seitenobjekt* zurück, was eine praktische Möglichkeit bietet, durch die Seiten zu blättern, ohne selbst die vorherige oder nächste Anfrage zu konstruieren.

ⓘ Mit der Methode *streamForward()* können alle Elemente über die Java 8 Stream API abgerufen und verarbeitet werden.

```
1.  IOperationClient operationClient = api.getOperationClient();
2.
3.  /* Every response will return a Page object with the specific Operation Response */
4.  Page<OperationResponse> operationsPage = operationClient.getOperations(new
    GetOperationsRequest());
5.
6.  /* Meta data of the current pagination */
7.  CollectionResponse<OperationResponse> response = operationsPage.getResponse();
8.  Pagination pagination = response.getPagination();
9.  int limit = pagination.getLimit();
10. int count = pagination.getCount();
11. Optional<Integer> offset = pagination.getOffset();
12. long total = pagination.getTotal();
13.
14. /* Retrieve the Elements of the current page */
15. List<OperationResponse> elements = operationsPage.getElements();
16.
17. try {
18. /* Retrieve the first specific page */
19. SpecificPage<OperationResponse, CollectionResponse<OperationResponse>> first =
    operationsPage.getFirstPage();
20. /* Retrieve the next specific page */
21. SpecificPage<OperationResponse, CollectionResponse<OperationResponse>> next =
    operationsPage.getNextPage();
22. /* Retrieve the previous specific page */
23. SpecificPage<OperationResponse, CollectionResponse<OperationResponse>> previous =
    operationsPage.getPreviousPage();
24. /* Retrieve the last specific page */
25. SpecificPage<OperationResponse, CollectionResponse<OperationResponse>> last =
    operationsPage.getLastPage();
26. } catch (PageNotAvailableException ex) {
27. LOGGER.error(ex.getMessage());
28. }
29.
30. /* Stream through all pages forward */
31. Stream<OperationResponse> operationForwardStream = operationsPage.streamForward();
32.
33. /* Stream through all pages backward */
34. Stream<OperationResponse> operationBackwardStream = operationsPage.streamBackward();
```

CURL

```
..literalinclude:: /../_code/curl/
```

⚠ Einige Ressourcen werden häufiger aktualisiert, was zu einem Kreislaufeffekt führen könnte, wenn eine Antwort durchgeblättert wird. Um dies zu vermeiden, wird der Versatzparameter hier nicht verwendet, sondern es werden zusätzliche Felder eingeführt.

2.1.3.3.1 Häufig aktualisierte Ressourcen

Jede Ressource, die häufig aktualisiert wird, kann nicht mit dem *Versatzparameter* nummeriert werden, weil womöglich neue Ressourceneinträge erzeugt werden, während die nächste Seite einer vorherigen Anfrage abgerufen wird. Dies würde mehrere Probleme verursachen, z. B. doppelte Einträge durch Verschieben der neuen Einträge an den Anfang der ersten Seite.

Um dies zu verhindern, wird für jede betroffene Ressource ein anderer Seitenmechanismus verwendet. Die folgenden Felder definieren bei Anfragen das Verhalten der Seitennummerierung:

paginationDirection
Die Seitennummerierungsrichtung legt fest, wie die Anfrage sortiert werden soll. Entweder *NEXT* oder *PREVIOUS* aus der Anfrage. Werte: *NEXT* (Voreinstellung), *PREVIOUS*

paginationTimestamp
Der erste oder letzte Zeitstempel der Seitennummerierung des jeweiligen Zeitraums.

paginationIdentifier
Eindeutige Bezeichnung für die Seitennummerierung für die Fälle, in denen der Zeitstempel nicht eindeutig ist, aber mehrfach vorhanden ist.

Bei jeder Antwort werden die folgenden zwei zusätzlichen Bezeichner gesetzt.

firstIdentifier
lastIdentifier

Beide Eigenschaften haben einen eindeutigen *paginationIdentifier* und *paginationTimestamp*, der von FORCE zum Zeitpunkt der Anfrage gesetzt wird.

2.1.3.4 Ressourcen einbetten

Java

Teilressourcen können wie folgt eingebettet und abgerufen werden. Im folgenden Beispiel werden speziell die Vorgänge für die Produktionsaufträge eingebettet.

```
1.      IProductionOrderClient productionOrderClient = api.getProductionOrderClient();
2.
3.      GetProductionOrdersRequest ordersRequest = new GetProductionOrdersRequest();
4.      ordersRequest.embed(new ProductionOrderEmbed().operations(true));
5.
6.      Page<ProductionOrderResponse> productionOrders =
    productionOrderClient.getProductionOrders(ordersRequest);
7.
8.      List<OperationPropertiesWSModel> operations = productionOrders
9.          .streamForward()
10.         .flatMap(productionOrderResponse -> productionOrderResponse
11.             .getOperations()
12.             .stream())
13.         .map(LinkEmbeddedWSModel::getEmbedded)
14.         .collect(Collectors.toList());
```

CURL

```
curl -X GET "http://$HOST:$PORT/ffwebservices/api/v2/$resource?embed=$embedableResource" \
    --header  "accept: application/json;charset=UTF-8" \
    --header  "authorization: Bearer $TOKEN"
```

2.1.4 Tutorials

Tutorial 1

Jeden Handarbeits- und Maschinenarbeitsplatz finden und auflisten.

Lösung:

Java

```
1.      final IWorkplaceClient workplaceClient = api.getWorkplaceClient();
2.
3.      try {
4.          final Page<WorkplaceResponse> workplaceResponsePage =
    workplaceClient.getWorkplaces(new GetWorkplacesRequest());
5.
6.          final Pagination pagination = workplaceResponsePage
7.              .getResponse()
8.              .getPagination();
9.
10.         final long totalWorkplaces = pagination.getTotal();
11.
12.         final Map<Boolean, List<WorkplaceResponse>> groups = workplaceResponsePage
13.             .streamForward()
14.             .collect(Collectors.partitioningBy(workplaceResponse -> workplaceResponse
15.                 .getProperties()
16.                 .isIsManualWorkplace()));
17.
18.         for (Map.Entry<Boolean, List<WorkplaceResponse>> entry : groups.entrySet()) {
19.
20.             final Boolean isManualWorkplace = entry.getKey();
21.             final List<WorkplaceResponse> responses = entry.getValue();
22.             final int sumWorkplaces = responses.size();
23.
24.             if (isManualWorkplace) {
25.                 System.out.println("Manual Workplaces: (" + sumWorkplaces + "/" +
    totalWorkplaces + ")");
26.             } else {
27.                 System.out.println("Machine Workplaces: (" + sumWorkplaces + "/" +
    totalWorkplaces + ")");
28.             }
29.
30.             responses.forEach(workplaceResponse -> {
31.                 final WorkplacePropertiesWSModel properties =
    workplaceResponse.getProperties();
32.
33.                 System.out.printf("\t %5s - %s%n", properties.getNumber(),
    properties.getDescription());
34.             });
35.
36.             System.out.println();
37.         }
38.     } catch (ForceAPIException e) {
39.         LOGGER.error(e.getMessage());
40.     }
```

Tutorial 2

Alle Handarbeitsplätze in jeder Kapazitätsgruppe finden und auflisten.

Lösung:

Java

```
1.      final IWorkplaceGroupClient workplaceGroupClient = api.getWorkplaceGroupClient();
2.      final IWorkplaceClient workplaceClient = api.getWorkplaceClient();
3.
4.      try {
5.
6.          final GetWorkplaceGroupsRequest workplaceGroupsRequest = new
    GetWorkplaceGroupsRequest();
7.          workplaceGroupsRequest.setType(WorkplaceGroupType.CAPACITY GROUP);
8.
9.          final Page<WorkplaceGroupResponse> workplaceGroupResponsePage =
    workplaceGroupClient.getWorkplaceGroups(workplaceGroupsRequest);
10.         final List<String> capacityGroupIds = workplaceGroupResponsePage
11.             .streamForward()
12.             .map(workplaceGroups -> workplaceGroups
13.                 .getProperties()
14.                 .getId())
15.             .collect(Collectors.toList());
16.
17.         if (capacityGroupIds.isEmpty()) {
18.             System.out.println("No Capacity Groups!");
19.             return;
20.         }
21.
22.         Map<String, List<WorkplaceResponse>> capacityGroupManualWorkplaces =
    capacityGroupIds
23.             .stream()
24.             .collect(Collectors.toMap(capacityGroupId -> capacityGroupId,
    capacityGroupId -> {
25.
26.                 GetWorkplacesRequest workplacesRequest = new GetWorkplacesRequest();
27.                 workplacesRequest.setWorkplaceGroupNumber(capacityGroupId);
28.                 List<WorkplaceResponse> collect = new ArrayList<>();
29.                 try {
30.                     collect.addAll(workplaceClient
31.                                         .getWorkplaces(workplacesRequest)
32.                                         .streamForward()
33.                                         .filter(workplace -> workplace
34.                                             .getProperties()
35.                                             .isIsManualWorkplace())
36.                                         .collect(Collectors.toList()));
37.                 } catch (ForceAPIException e) {
38.                     LOGGER.error(e.getMessage());
39.                 }
40.                 return collect;
41.             }));
42.
43.         System.out.println("Manual Workplaces for");
44.         capacityGroupManualWorkplaces.forEach((capacityGroupId, workplaceResponses) -
    > {
45.             System.out.printf("\tCapacity Group [%s]\n", capacityGroupId);
46.             workplaceResponses
47.                 .stream()
48.                 .map(WorkplaceResponse::getProperties)
49.                 .forEach(p -> System.out.printf("\t\tWorkplace %s [%s]\n",
    p.getNumber(), p.getId()));
50.         });
51.
52.     } catch (ForceAPIException e1) {
53.         e1.printStackTrace();
54.     }
```

/../_code/java/src/main/java/com/forcam/apps/ManualLaborWorkplacesInCapacityGroups.java

Tutorial 3

Alle Produktionsaufträge der aktuellen Woche suchen, für die alle Vorgänge abgeschlossen sind.

Lösung:

Java

```
1.          IProductionOrderClient productionOrderClient = api.getProductionOrderClient();
2.
3.          Interval currentWeek = Interval.getWeekInterval(1, 0);
4.
5.          try {
6.              GetProductionOrdersRequest ordersRequest = new
    GetProductionOrdersRequest();
7.              ordersRequest.setStartDate(currentWeek.getStartDate());
8.              ordersRequest.setEndDate(currentWeek.getEndDate());
9.              ordersRequest.embed(new ProductionOrderEmbed().operations(true));
10.             Page<ProductionOrderResponse> orderResponses =
    productionOrderClient.getProductionOrders(ordersRequest);
11.
12.             List<ProductionOrderResponse> productionOrderResponses =
    orderResponses.getElements();
13.
14.             List<ProductionOrderResponse> completedProductionOrdersResponses =
    productionOrderResponses
15.                 .stream()
16.                 .filter(productionOrderResponse -> productionOrderResponse
17.                     .getOperations()
18.                     .stream()
19.                     .map(LinkEmbeddedWSModel::getEmbedded)
20.                     .noneMatch(operationProperties ->
    operationProperties.getOperationPhaseId() != OperationPhase.COMPLETED))
21.                 .collect(Collectors.toList());
22.
23.             printProductionOrders(completedProductionOrdersResponses);
24.
25.         } catch (ForceAPIException e) {
26.             LOGGER.error(e.getMessage());
27.         }
```

/../_code/java/src/main/java/com/forcam/apps/CompletedProductionOrders.java

Tutorial 4

Herausfinden, in welchen Betriebszuständen sich alle Arbeitsplätze befinden und wie viel Prozent der Arbeitsplätze sich im Betriebszustand Produktion befinden.

Lösung:

Java

```
1.          final IWorkplaceClient workplaceClient = api.getWorkplaceClient();
2.
3.          try {
4.              final GetWorkplacesRequest workplacesRequest = new GetWorkplacesRequest();
5.              final Page<WorkplaceResponse> workplaceResponsePage =
    workplaceClient.getWorkplaces(workplacesRequest);
6.              final Pagination pagination = workplaceResponsePage
7.                  .getResponse()
8.                  .getPagination();
9.
10.             final long totalNumberOfWorkplaces = pagination.getTotal();
11.
12.             workplaceResponsePage
13.                 .streamBackward()
14.                 .map(WorkplaceResponse::getProperties)
15.                 .forEach(workplace -> System.out.printf("Workplace %s: %s%n",
16.                                                         workplace.getNumber(),
17.                                                         workplace
18.                                                             .getOperatingState()
19.                                                             .getDescription()));
20.
21.             final long numberOfWorkplacesInProduction = workplaceResponsePage
22.                 .streamForward()
23.                 .map(WorkplaceResponse::getProperties)
24.                 .filter(properties -> properties
25.                     .getOperatingState()
26.                     .getWorkplaceStateId()
27.                     .equals(WorkplaceState.PRODUCTION))
28.                 .count();
29.
30.             float workplacesInProduction = ((float) numberOfWorkplacesInProduction /
    (float) totalNumberOfWorkplaces) * 100;
31.             System.out.printf("Workplaces in Production %.2f %%%n",
    workplacesInProduction);
32.
33.         } catch (ForceAPIException e) {
34.             LOGGER.error(e.getMessage());
35.         }
```

/../_code/java/src/main/java/com/forcam/apps/OperatingStatesOfWorkplaces.java

Tutorial 5

Ermitteln in welcher Vorgangsphase jeder Vorgang gerade ist und den Prozentsatz berechnen, wie viele laufende Vorgänge sich jeweils in der Phase Rüsten bzw. in der Phase Bearbeitung befinden.

Lösung:

Java

```
1.          final IOperationClient operationClient = api.getOperationClient();
2.
3.          try {
4.              final GetOperationsRequest operationsRequest = new GetOperationsRequest();
5.              final Page<OperationResponse> operationResponsePage =
    operationClient.getOperations(operationsRequest);
6.
7.              final Pagination pagination = operationResponsePage
8.                  .getResponse()
9.                  .getPagination();
10.
11.             final long totalNumberOfOperations = pagination.getTotal();
12.             System.out.println("Number of Operations: " + totalNumberOfOperations);
13.
14.             final Map<OperationPhase, List<OperationResponse>> groups =
    operationResponsePage
15.                 .streamForward()
16.                 .collect(Collectors.groupingBy(e -> e
17.                     .getProperties()
18.                     .getOperationPhaseId()));
19.
20.             int numberOfOperationsInSetup = 0;
21.             int numberOfOperationsInProcessing = 0;
22.
23.             if (groups.containsKey(OperationPhase.SETUP)) {
24.                 numberOfOperationsInSetup = groups
25.                     .get(OperationPhase.SETUP)
26.                     .size();
27.             }
28.             if (groups.containsKey(OperationPhase.PROCESSING)) {
29.                 numberOfOperationsInProcessing = groups
30.                     .get(OperationPhase.PROCESSING)
31.                     .size();
32.             }
33.
34.             float percentageOperationsInProcessing = ((float)
    numberOfOperationsInProcessing / totalNumberOfOperations) * 100;
35.             float percentageOperationsInSetup = ((float) numberOfOperationsInSetup /
    totalNumberOfOperations) * 100;
36.             System.out.printf("Percentage of Operations in Phase Setup: %.2f %%%n",
    percentageOperationsInSetup);
37.             System.out.printf("Percentage of Operations in Phase Processing: %.2f
    %%%n", percentageOperationsInProcessing);
38.
39.         } catch (ForceAPIException e) {
40.             LOGGER.error(e.getMessage());
41.         }
```

/../_code/java/src/main/java/com/forcam/apps/OperationPhasesOfOperations.java

2.2 Schichten und geplante Betriebszeiten von Arbeitsplätzen

2.2.1 Domänenspezifisches Wissen

2.2.1.1 Arbeitsplatzschichten

Der Begriff Schichtarbeit beschreibt eine Arbeitsstruktur, bei der Mitarbeiter nacheinander und nach einem bestimmten Zeitplan an der gleichen Tätigkeit arbeiten, so dass sie innerhalb eines bestimmten Zeitrahmens zu unterschiedlichen Zeiten ihre Arbeit verrichten. Ein Betrieb setzt Schichtarbeit ein, wenn es aus verschiedenen Gründen notwendig ist, außerhalb der üblichen Tagesarbeitszeit zu arbeiten oder wenn Betriebs- oder Bereitschaftsdienste außerhalb der üblichen Tagesarbeitszeit erforderlich sind.

Für Arbeitsplätze werden Schichten definiert, z. B. im Dreischichtbetrieb sind dies meist Frühschicht, Spätschicht und Nachtschicht. Dieses Konzept beschreibt ein Schichtmodell.

Wenn eine der Schichten an einem bestimmten Tag ausfallen soll, wird die Schicht als Leerlaufzeit oder arbeitsfreie Schicht gekennzeichnet. Die anderen Schichten sind Arbeitsschichten. Innerhalb einer Arbeitsschicht können sogenannte Schichtpausen definiert werden, während derer die Produktion angehalten wird.

2.2.1.2 Geplante Wartungszeiten

Dies sind Zeiten, in denen Maschinenarbeitsplätze gewartet werden. Handarbeitsplätze sind davon nicht betroffen. Das Ziel geplanter Wartungszeiten ist es, Störungen zu reduzieren, damit Maschinen nicht unerwartet ausfallen, was wiederum eine außerplanmäßige Wartung auslösen würde. Diese Zeiten finden in regelmäßigen Wartungsintervallen statt.

2.2.1.3 Geplante Betriebszeiten

Die planmäßige Betriebszeit ist der Zeitraum, in dem ein Arbeitsplatz laut Schichtplanung und Wartungsplanung in Produktion sein soll. Dazu gehören alle Arbeitsschichten abzüglich der Schichtpausen und der geplanten Wartung.

2.2.2 Das digitale Abbild in FORCE Bridge API

Die oben genannten Objekte befinden sich unter den Teilressourcen von Arbeitsplätzen bzw. einem bestimmten Arbeitsplatz.

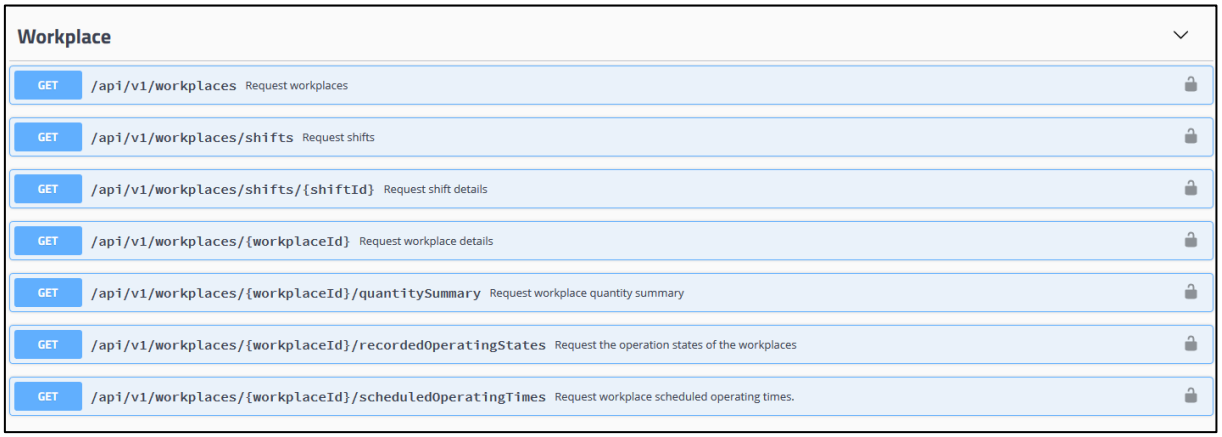

Bild 3: Arbeitsplatz-Ressourcen, wie in der Swagger UI **dargestellt.**

GET workplaces/shifts
Schichten eines Arbeitsplatzes anfragen.

GET workplaces/shifts/{shiftId}
Details für eine bestimmte Schicht anfragen.

GET workplaces/{workplaceId}/scheduledOperatingTimes
Arbeitsplatz geplante Betriebszeiten anfragen.

GET workplace/{workplaceId}/scheduledMaintenanceTimes
Anfrage der geplanten Wartungszeiten eines Arbeitsplatzes.

2.2.3 Tutorials

Tutorial 1

Die geplanten Schichten und Wartungstermine für die nächsten zwei Wochen abfragen. Danach, die geplanten Betriebszeiten berechnen. Anschließend die berechneten Zeiten mit den Zeiten aus der Schnittstelle für geplante Betriebszeiten vergleichen.

Lösung:

Java

```
1.          IWorkplaceClient workplaceClient = api.getWorkplaceClient();
2.          Interval nextTwoWeeks = Interval.getWeekInterval(2, 2);
3.
4.          /* Retrieve all workplaces */
5.          Page<WorkplaceResponse> workplaceResponsePage = null;
6.
7.          try {
8.              GetWorkplacesRequest workplacesRequest = new GetWorkplacesRequest();
9.              workplaceResponsePage = workplaceClient.getWorkplaces(workplacesRequest);
10.
11.         } catch (ForceAPIException e) {
12.             LOGGER.error(e.getMessage());
13.         }
14.
15.         List<WorkplaceResponse> workplaceResponses = workplaceResponsePage
16.             .streamForward()
17.             .collect(Collectors.toList());
18.
19.         /* Cache the workplace UUIDs */
20.         List<String> workplaceIDs = workplaceResponses
21.             .stream()
22.             .map(WorkplaceResponse::getProperties)
23.             .map(WorkplacePropertiesWSModel::getId)
24.             .collect(Collectors.toList());
25.
26.         /* Get the shifts of all workplaces*/
27.         GetWorkplaceShiftsRequest workplaceShiftsRequest = new
    GetWorkplaceShiftsRequest();
28.         workplaceShiftsRequest.setStartDate(nextTwoWeeks.getStartDate());
29.         workplaceShiftsRequest.setEndDate(nextTwoWeeks.getEndDate());
30.
31.         Page<WorkplaceShiftResponse> shiftResponsesPage = null;
32.         try {
33.             shiftResponsesPage = workplaceClient.getShifts(workplaceShiftsRequest);
34.         } catch (ForceAPIException e) {
35.             LOGGER.error(e.getMessage());
36.         }
37.         final List<WorkplaceShiftResponse> shiftResponses = shiftResponsesPage
38.             .streamForward()
39.             .collect(Collectors.toList());
40.
41.         for (WorkplaceShiftResponse shiftResponse : shiftResponses) {
42.
43.         }
44.
45.
46.         /* For each workplace get the scheduled maintenance dates */
47.         final Map<String, List<TimePeriodWSModel>> collect = workplaceIDs
48.             .stream()
49.             .collect(Collectors.toMap(k -> k, v -> {
50.                 GetWorkplaceScheduledTimesRequest scheduledTimesRequest = new
    GetWorkplaceScheduledTimesRequest(v);
51.                 scheduledTimesRequest.setStartDate(nextTwoWeeks.getStartDate());
52.                 scheduledTimesRequest.setEndDate(nextTwoWeeks.getEndDate());
53.
54.                 WorkplaceScheduledMaintenanceTimesCollectionPage
    scheduledMaintenanceTimesPage = null;
55.                 try {
56.                     scheduledMaintenanceTimesPage =
    workplaceClient.getWorkplaceScheduledMaintenanceTimes(scheduledTimesRequest);
```

```
57.                 } catch (ForceAPIException e) {
58.                     LOGGER.error(e.getMessage());
59.                 }
60.                 return scheduledMaintenanceTimesPage
61.                     .streamForward()
62.                     .collect(Collectors.toList());
63.
64.             }));
65.
66.         /* For each workplace get scheduled operating times */
67.         Map<String, List<TimePeriodWSModel>> scheduledOperatingTimes = workplaceIDs
68.             .stream()
69.             .collect(Collectors.toMap(k -> k, v -> {
70.                 GetWorkplaceScheduledTimesRequest scheduledTimesRequest = new
    GetWorkplaceScheduledTimesRequest(v);
71.                 scheduledTimesRequest.setStartDate(nextTwoWeeks.getStartDate());
72.                 scheduledTimesRequest.setEndDate(nextTwoWeeks.getEndDate());
73.
74.                 WorkplaceScheduledOperatingTimesCollectionPage
    scheduledOperatingTimesPage = null;
75.                 try {
76.                     scheduledOperatingTimesPage =
    workplaceClient.getScheduledOperatingTimes(scheduledTimesRequest);
77.                 } catch (ForceAPIException e) {
78.                     LOGGER.error(e.getMessage());
79.                 }
80.                 return scheduledOperatingTimesPage
81.                     .streamForward()
82.                     .collect(Collectors.toList());
83.
84.             }));
```

Tutorial 2

Die geplanten Schichten und geplanten Wartungsdaten der letzten zwei Wochen abfragen. Dann die geplanten Betriebszeiten auf Basis der abgefragten Werte berechnen. Anschließend die Ergebnisse mit den Werten aus der Schnittstelle der geplanten Betriebszeiten vergleichen.

ⓘ Die geplanten Betriebszeiten sollten immer identisch sein.

Lösung:

Java

```
1.          final IWorkplaceClient workplaceClient = api.getWorkplaceClient();
2.
3.          try {
4.              final Interval previousTwoWeeks = Interval.getWeekInterval(2, -1);
5.
6.              final GetWorkplacesRequest workplacesRequest = new GetWorkplacesRequest();
7.              final Page<WorkplaceResponse> workplaceResponsePage =
    workplaceClient.getWorkplaces(workplacesRequest);
8.              workplaceResponsePage
9.                  .streamForward()
10.                 .map(WorkplaceResponse::getProperties)
11.                 .map(WorkplacePropertiesWSModel::getId)
12.                 .forEach(workplaceId -> {
13.                     try {
14.
15.                         /* Retrieve the scheduled shifts*/
16.                         final GetWorkplaceShiftsRequest workplaceShiftsRequest = new
    GetWorkplaceShiftsRequest();
17.                         workplaceShiftsRequest.setWorkplaceId(workplaceId);
18.
    workplaceShiftsRequest.setPaginationTimestamp(previousTwoWeeks.getEndDate());
19.
    workplaceShiftsRequest.setPaginationDirection(PaginationDirection.PREVIOUS);
20.
21.                         final Page<WorkplaceShiftResponse> shifts =
    workplaceClient.getShifts(workplaceShiftsRequest);
22.                         final Map<String, ShiftWSModel> workplaceScheduledShifts =
    shifts
23.                             .streamForward()
24.                             .map(WorkplaceShiftResponse::getProperties)
25.                             .collect(Collectors.toMap(p -> workplaceId,
    CreateWorkplaceShiftPropertiesWSModel::getShift));
26.
27.                         /* Retrieve the scheduled maintenance dates */
28.                         final GetWorkplaceScheduledTimesRequest scheduledTimesRequest
    = new GetWorkplaceScheduledTimesRequest(workplaceId);
29.
    scheduledTimesRequest.setStartDate(previousTwoWeeks.getStartDate());
30.
    scheduledTimesRequest.setEndDate(previousTwoWeeks.getEndDate());
31.                         final WorkplaceScheduledMaintenanceTimesCollectionPage
    scheduledMaintenancePages = workplaceClient.getWorkplaceScheduledMaintenanceTimes(
32.                             scheduledTimesRequest);
33.                         final Map<String, TimePeriodWSModel>
    workplaceScheduledMaintenance = scheduledMaintenancePages
34.                             .streamForward()
35.                             .collect(Collectors.toMap(k -> workplaceId, v -> v));
36.
37.                     } catch (ForceAPIException e) {
38.                         LOGGER.error(e.getMessage());
39.                     }
40.
41.                 });
42.
43.         } catch (ForceAPIException e) {
44.             LOGGER.error(e.getMessage());
45.         }
```

2.3 Vorgangsabläufe und Termine eines Produktionsauftrags

2.3.1 Domänenspezifisches Wissen

2.3.1.1 Vorgangsabläufe und Parallelisierung

Die Vorgänge eines Produktionsauftrags können sequenziell oder parallelisiert abgearbeitet werden.

Bei einem sequenziellen Ablauf wird beim Abschluss eines Vorgangs der Folgevorgang erst dann gestartet, wenn der Vorgang vollständig abgeschlossen wurde.

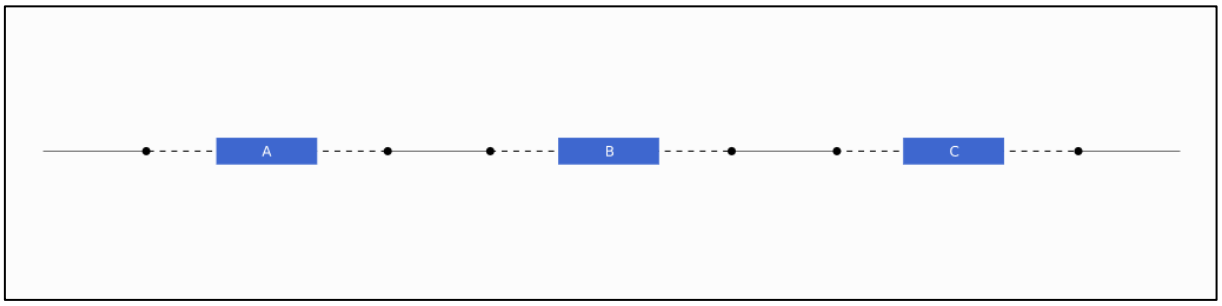

Bild 4: Sequenzieller Betriebsablauf

Häufig kann ein Vorgang (B) erst gestartet werden, wenn der vorhergehende Vorgang (A) abgeschlossen ist, weil er eventuell die Komponenten herstellt, die im nachfolgenden Vorgang benötigt werden.
Auch voneinander unabhängige Vorgänge können parallelisiert abgeschlossen werden, sofern genügend Kapazitäten vorhanden sind.

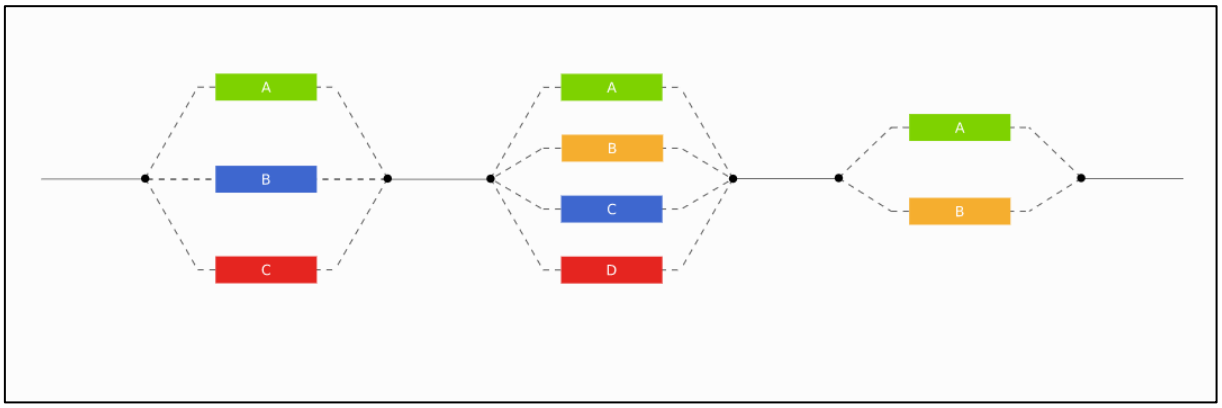

Bild 5: Parallelisierter Betriebsablauf

2.3.1.2 Planung

Die Planung findet bei der Einplanung eines neuen Produktionsauftrags statt und wird im ERP-System hinterlegt. Mehrere Faktoren spielen bei der Planung eine Rolle.

Das System prüft, ob zusätzliche Komponenten extern bestellt werden müssen und, ob der Liefertermin sowie das Freigabedatum verschoben bzw. berücksichtigt werden müssen.

Die Ecktermine der jeweiligen Bearbeitung müssen unter Berücksichtigung des Sicherheitsabstandes vor der Produktion[3] und des Sicherheitsabstandes nach der Produktion[4] festgelegt werden. Diese Termine bestimmen den maximalen Zeitrahmen für die Produktion.

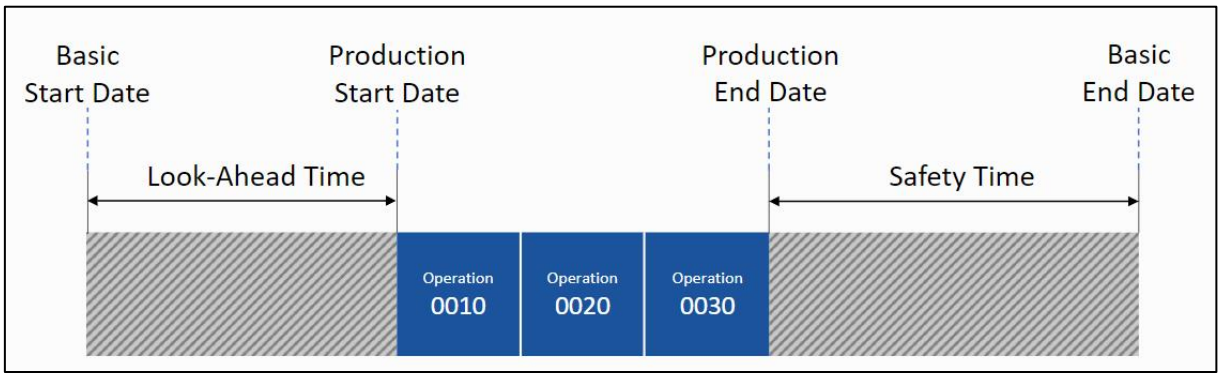

Bild 6: Maximalen Zeitrahmen für die Produktion festlegen

2.3.1.3 Durchlaufzeit planen

Die Planung der Durchlaufzeit ist die Festlegung von Start- und Endterminen der einzelnen Vorgänge eines Produktionsauftrags. Basierend auf den Eckterminen wird entweder vorwärts- oder rückwärtsgerichtet geplant.

Die früheste Startzeit (/EST) und die früheste Endzeit (/EET) werden auf der Basis des Auftragsstartdatums vorwärts geplant.

Die späteste Startzeit (/LST) und die späteste Endzeit (/LET) werden auf Basis des Auftragsendtermins rückwärts geplant.

Das folgende Beispiel erklärt, wie die kontinuierliche Planung vorwärts bzw. rückwärts ermittelt wird:

Ein Auftrag mit der Nummer 4711 hat drei Vorgänge. Der Auftragsstarttermin ist *Sonntag 22:00 Uhr* und der Auftragsendtermin ist der folgende *Dienstag um 12:00 Uhr*.

Schichtpausen sind nur am Montag von *12:00 Uhr* bis *14:00 Uhr* und von *22:00 Uhr* bis *24:00 Uhr*.

[3] Die Vorgriffszeit ist ein Startpuffer für mögliche Verzögerungen bei der Bereitstellung und Kapazitätsengpässen zur Anpassung der Produktionstermine eines Auftrags.

[4] Die Sicherheitszeit ist ein Endpuffer, um unvorhergesehene Störungen im Produktionsprozess abzufangen.

Die Vorgänge sind wie folgt definiert:

Vorgang	4711 0010	4711 0020	4711 0030
Sollmenge	4 pc(s)	4 pc(s)	4 pc(s)
Sollzeit pro Einheit	1 h	2 h	2 h
Sollrüstzeit	2 h	4 h	2 h
Soll-Abrüstzeit	2 h		
Liegezeit	2 h		
Wartezeit			
Mindestvorlaufmenge		2 pc(s)	

ⓘ Da es für den Vorgang 4711 0020 eine Mindestvorlaufmenge gibt, muss eine Mindestvorlaufzeit eingehalten werden.

Die minimale Offset-Zeit wird wie folgt berechnet:

$$Minimale\ Offset\ Zeit = Mindestvorlaufmenge \times\ Sollzeit\ pro\ Einheit$$

So wird für das Beispiel die minimale Offset-Zeit wie folgt berechnet:

$$Minimale\ Offset\ Zeit = 2 * 2\ h = 4\ h$$

⚠ Pausen oder Freischichten verlängern nur die Ausführungszeiten (Rüsten, Bearbeiten, Abrüsten) und damit auch die Durchlaufzeit, nicht aber die Übergangszeiten (Wartezeit, Liegezeit, Transportzeit).

Die Vorwärtsterminierung basiert auf die Eckstartzeit und die Rückwärtsterminierung auf die Eckendzeit. Daraus ergeben sich die frühesten Zeiten für die Vorwärtsterminierung und die spätesten Zeiten für die Rückwärtsterminierung.

2.3.1.3.1 Ergebnisse

Vorgang	1000 0010	1000 0020	1000 0030
Früheste Startzeit	So, 22:00 Uhr	Mo, 10:00 Uhr	Mo, 18:00 Uhr
Früheste Endzeit	Mo, 06:00 Uhr	Di, 02:00 Uhr	Di, 06:00 Uhr
Späteste Startzeit	Mo, 04:00 Uhr	Mo, 16:00 Uhr	Di, 00:00 Uhr
Späteste Endzeit	Mo, 12:00 Uhr	Di, 08:00 Uhr	Di, 12:00 Uhr

2.3.1.3.1.1 Vorwärtsterminierung

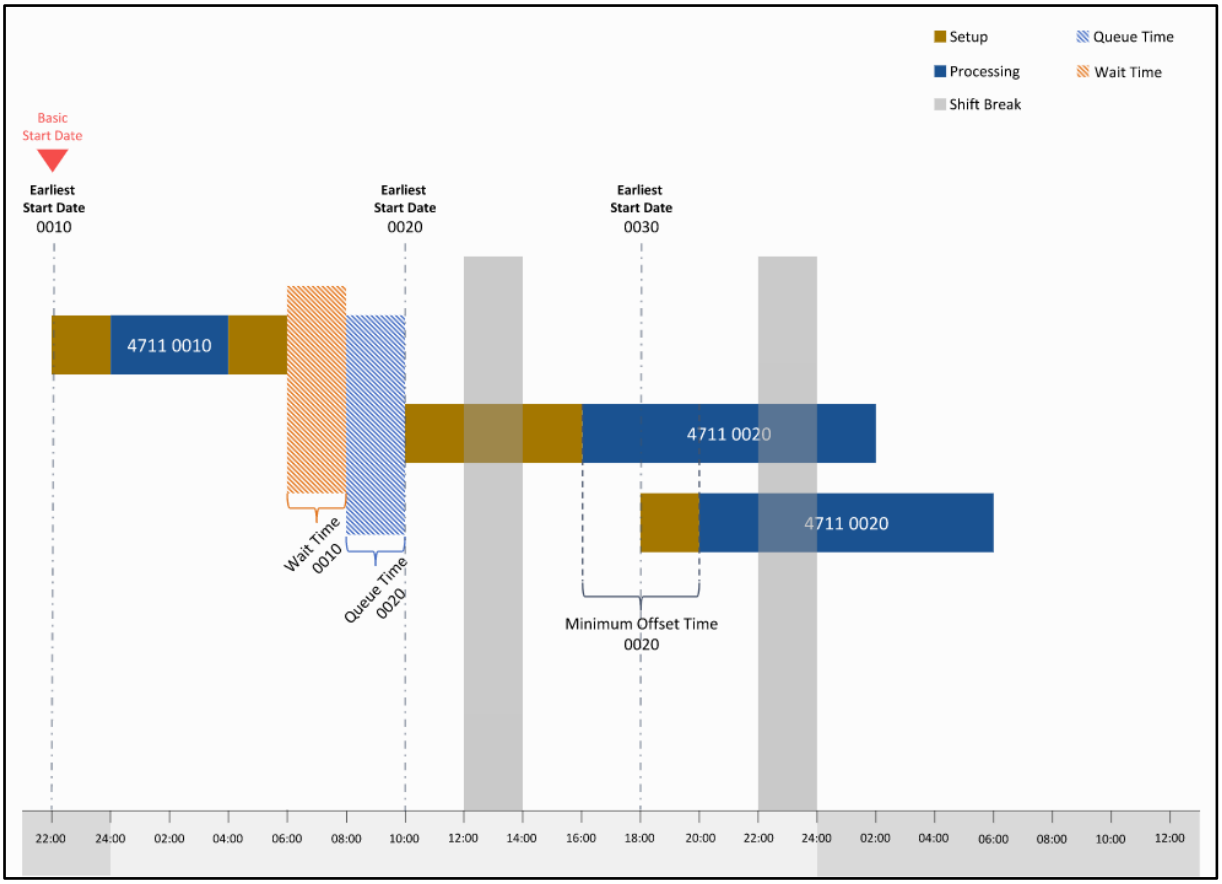

Bild 7: Vorwärtsterminierung

Geplante Startzeit mit Vorwärtsterminierung	So, 22:00 Uhr
Geplante Endzeit mit Vorwärtsterminierung	Di, 06:00 Uhr

2.3.1.3.1.2 Rückwärtsterminierung

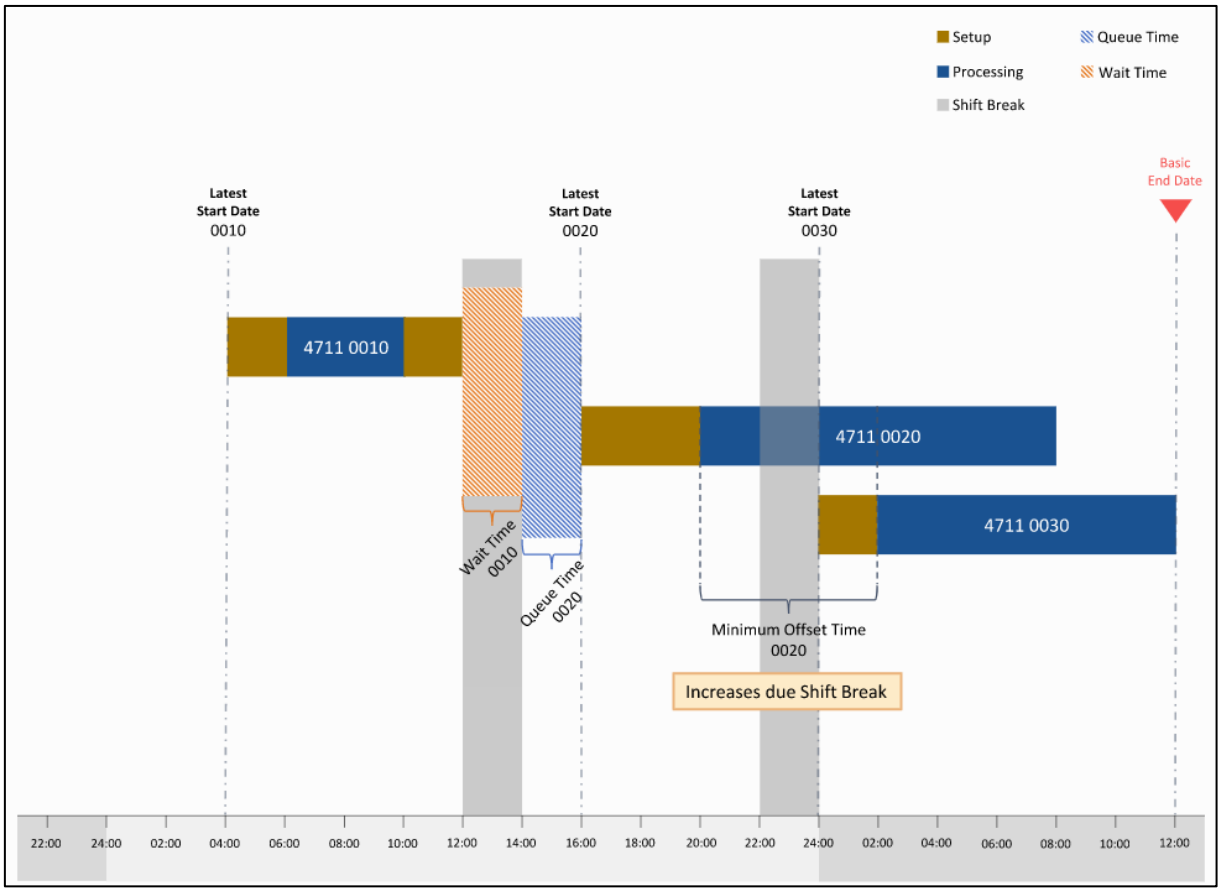

Bild 8: Rückwärtsterminierung

Geplante Startzeit mit Rückwärtsterminierung	Mo, 04:00 Uhr
Geplante Endzeit mit Rückwärtsterminierung	Di, 12:00 Uhr

2.3.2 Das digitale Abbild in FORCE Bridge API

GET productionOrders/{productionOrderId}
Details eines Produktionsauftrags abfragen.

GET operations/{operationId}/scheduledDates
Geplante Terminierung eines Vorgangs abfragen.

GET productionOrders/{productionOrderId}/operationSequence
Alle Vorgänge eines Produktionsauftrages sowie deren Vorgänger und Nachfolger abfragen.

ⓘ Alle Zeitformate entsprechen der ISO 8601.

2.3.3 Tutorials

2.3.3.1 Geplante Terminierung eines Vorgangs abfragen

Lösung:

Java

```
1.  GetOperationScheduledDatesRequest scheduledDatesRequest = new
    GetOperationScheduledDatesRequest(operationId);
2.          OperationScheduledDatesResponse operationScheduledDatesResponse =
    operationClient.getScheduledDates(scheduledDatesRequest);
```

/../_code/java/src/test/java/com/forcam/usage/operations/ScheduledDatesTest.java

2.4 Vorgang Teilressourcen mit Routing-Informationen

Die Werte aus dem Routing können in den folgenden Ressourcen gefunden werden:

GET operations/{operationId}/specification

Enthält Zielwerte aus dem Routing. Die folgenden Zielwerte können gefunden werden:

- Bearbeitungszeit
- Rüstzeit
- Abrüstzeit
- Wartezeit
- Liegezeit
- Transportzeit
- Minimale Transportmenge

⚠ Die Sollrüstzeit und Sollabrüstzeit eines Vorgangs kann gleich Null sein. Somit kann der Vorgang nur aus Bearbeitungsphasen bestehen. FORCE Bridge stellt sicher, dass die Soll-Bearbeitungszeit entweder durch die Sollmenge oder die Sollhubzeit richtig berechnet wird.

GET operations/{operationId}/productionResourceTools
Werkzeuge und Produktionsressourcen, die für die Ausführung des Vorgangs benötigt werden.

GET operations/{operationId}/components
Komponenten, die für die Ausführung des Vorgangs benötigt werden.

GET operations/{operationId}/materialCharacteristics
Materialeigenschaften, die durch den Vorgang erzeugt werden sollen.

GET operations/{operationId}/alternativeWorkplaces
Arbeitsplätze, an denen der Vorgang ausgeführt werden kann. Dies sind die Arbeitsplätze in der zugeordneten Kapazitätsgruppe, wenn für das zu produzierende Material keine Fertigungsvarianten definiert sind.

ⓘ Außer der Sollmenge in der Spezifikationsressource werden alle diese Werte aus dem vom ERP-System erzeugten Arbeitsplan des entsprechenden Fertigungsauftrags übernommen.

2.4.1 Tutorials

2.4.1.1 Spezifikation der planbaren Vorgänge abrufen

Lösung:

Java

```
1.          IOperationClient operationClient = api.getOperationClient();
2.
3.          GetOperationsRequest operationsRequest = new GetOperationsRequest();
4.          operationsRequest.setIsPlannable(true);
5.          operationsRequest.embed(new OperationEmbed().specification(true));
6.
7.          Page<OperationResponse> operationResponsePage =
    operationClient.getOperations(operationsRequest);
8.
9.          operationResponsePage
10.             .streamForward()
11.             .forEach(operationResponse -> {
12.                 OperationPropertiesWSModel properties =
    operationResponse.getProperties();
13.                 OperationSpecificationPropertiesWSModel routingDetails =
    operationResponse
14.                     .getSpecification()
15.                     .getEmbedded();
16.                 /* ... */
17.             });
```

/../_code/java/src/test/java/com/forcam/usage/operations/EmbedOperationEntriesTest.java

CURL

```
curl -X GET
    "http://$HOST:$PORT/ffwebservices/api/v2/operations?embed=specification&isPlannable=tru
    e&limit=100&offset=0"
    --header "accept: application/json;charset=UTF-8"
    --header "authorization: Bearer $TOKEN"
```

2.4.1.2 Die erforderlichen Komponenten der in Bearbeitung stehenden Vorgangs abrufen

Lösung:

Java

```
1.         IOperationClient operationClient = api.getOperationClient();
2.          GetOperationsRequest operationsRequest = new GetOperationsRequest();
3.          operationsRequest.setOperationPhaseId(OperationPhase.PROCESSING);
4.
5.          Page<OperationResponse> operationResponsePage =
    operationClient.getOperations(operationsRequest);
6.
7.          List<OperationResponse> operationResponses = operationResponsePage
8.              .streamForward()
9.              .collect(Collectors.toList());
10.         Map<OperationResponse, List<OperationComponentWSModel>> operationComponentsMap
    = new HashMap<>();
11.         for (OperationResponse operationResponse : operationResponses) {
12.             String operationId = operationResponse
13.                 .getProperties()
14.                 .getId();
15.             GetOperationComponentsRequest componentsRequest = new
    GetOperationComponentsRequest(operationId);
16.             OperationComponentCollectionPage operationComponentResponses =
    operationClient.getOperationComponents(componentsRequest);
17.             List<OperationComponentWSModel> operationComponents =
    operationComponentResponses
18.                 .streamForward()
19.                 .collect(Collectors.toList());
20.             operationComponentsMap.put(operationResponse, operationComponents);
21.         }
```

/../_code/java/src/test/java/com/forcam/usage/operations/components/ComponentsTest.java

CURL

```
curl -X GET
    "http://$HOST:$PORT/ffwebservices/api/v2/operations?embed=components&operationPhaseId=P
    ROCESSING
    --header "accept: application/json;charset=UTF-8"
    --header "authorization: Bearer $TOKEN"
```

2.5 Maschinen- und Betriebsdatenerfassung und deren Leistungskennzahlen

Die Maschinendatenerfassung und die Betriebsdatenerfassung erfassen Istwerte, Zeiten und Mengen, die während des Fertigungsprozesses auftreten. Die Erfassung kann direkt an der Maschine oder durch einen Mitarbeiter am Shopfloor Terminal erfolgen. Das Shopfloor Terminal ist die zentrale Informationsquelle für die Mitarbeiter. Hier werden die auszuführenden Vorgänge angezeigt und nach ihrer geplanten Startzeit geordnet. Auch die zugehörigen Dokumente zu jedem Vorgang können hier angezeigt werden. Außerdem wird das Shopfloor Terminal zur Betriebsdatenerfassung genutzt, da hier Vorgänge und Mitarbeiter an- und abgemeldet werden. Darüber hinaus kann der Maschinenbediener alle Informationen über Betriebszustände und Qualitätsangaben der Ausgangsmenge zur Verfügung stellen, wenn diese nicht automatisch von der Maschine oder bei einem Handarbeitsplatz ausgelesen werden können.

2.5.1 Domänenspezifisches Wissen

2.5.1.1 Leistungskennzahlen

Leistungskennzahlen *(Key Performance Indicators KPI)* beschreiben den Anteil der Verluste einer Maschine, eines Prozesses oder geben Auskunft über das Optimierungspotenzial von Maschinen bzw. Prozessen. In diesem Kapitel werden speziell die folgenden zwei Informationssysteme untersucht.

Die Gesamtanlageneffektivität *(Overall Equipment Effectiveness (OEE)* betrachtet Verfügbarkeits-, Leistungs- und Qualitätsverluste und kann bei unbedeutenden Werten auf ungenutzte Produktionskapazitäten hinweisen, was einer Verschwendung von Ressourcen gleichkommt.

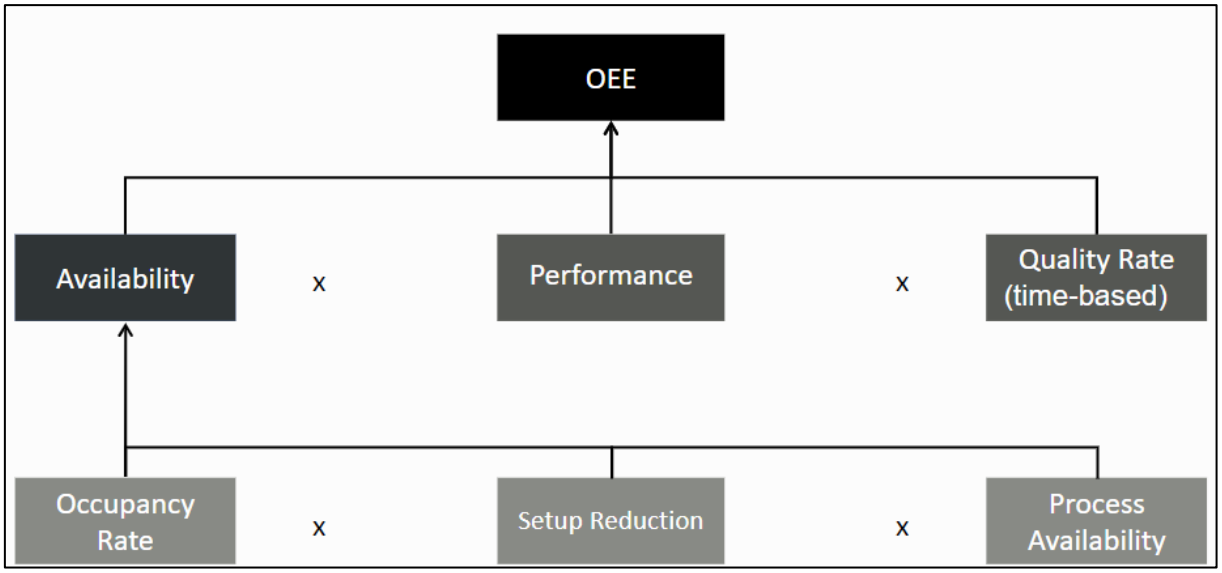

Bild 9: Overall Equipment Effectiveness

Die Gesamtprozesseffizienz *Overall Process Efficiency (OPE)* ist eine produktspezifische Kennzahl. Ein niedriger OPE weist dabei auf vermeidbare Kapitalbindungskosten[5] hin.

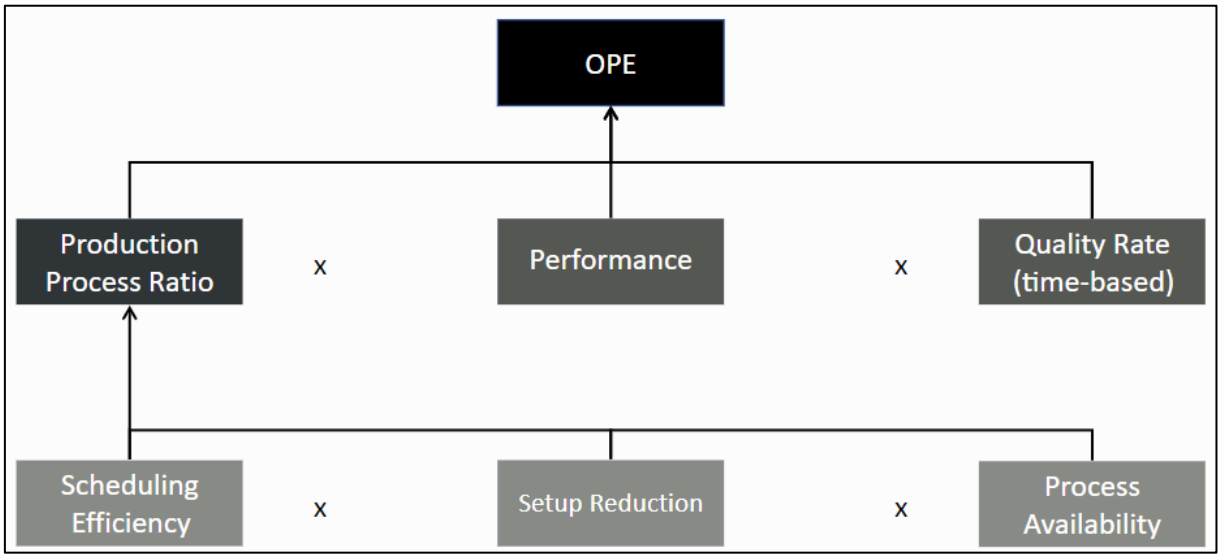

Bild 10: Overall Process Efficiency

2.5.1.2 Laufzeiten

Wie im Abschnitt 2.4 erläutert, gibt es eine Reihe von Sollwerten für die einzelnen Vorgänge eines Produktionsauftrags. Zum Beispiel in Bezug auf ihre Rüst- und Bearbeitungszeiten. Während der Auftragsbearbeitung ist es jedoch möglich, dass diese Sollwerte nicht erreicht oder eingehalten werden können. Die Istwerte werden von der IoT-Plattform auf Basis der Maschinen- und Betriebsdatenerfassung erfasst.

2.5.1.3 Unterschied zwischen Vorgangsphasen und Betriebszuständen

Das folgende Beispiel verdeutlicht den Unterschied zwischen Betriebsphase und Betriebszustand.

Ein Vorgang ist einem Arbeitsplatz zugeordnet. Um die Bearbeitung zu starten, meldet sich der Mitarbeiter am Shopfloor Terminal an und schaltet - falls zutreffend - auf *Rüsten*. Damit wird die Vorgangsphase Rüsten erfasst. Nachdem das Rüsten abgeschlossen ist, schaltet der Werker den Vorgang auf *Bearbeitung* um. Die Betriebsphase *Rüsten* ist damit abgeschlossen und wechselt in die Phase *Bearbeiten*. Start- und Endzeitpunkt der Phasen werden im System festgehalten.

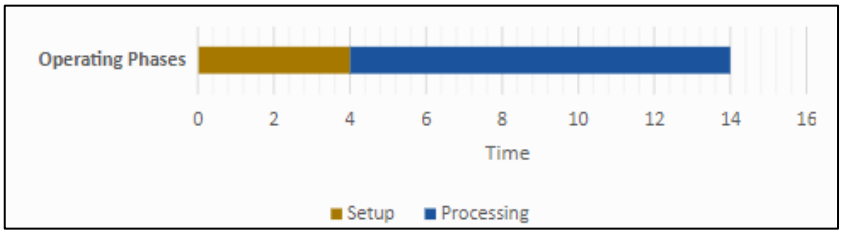

Bild 11: Erfasste Vorgangsphasen

[5] Kapitalbindungskosten sind Kosten für das im Unternehmen gebundene Anlage- und Umlaufvermögen, das deshalb nicht anderswo gewinnbringend eingesetzt werden kann.

Parallel zum Start der Vorgangsphase *Rüsten* wird auch der Betriebszustand *Rüsten* aufgezeichnet. Beim fließenden Wechsel in die Phase *Bearbeitung* wird der Betriebszustand *Produktion* erfasst. Der Arbeitsplatz befindet sich nun im Zustand *Produktion*. Wenn nach einiger Zeit die Maschine aus unbekannten Gründen angehalten wird, wird der Betriebszustand *Stillstand unbegründet* protokolliert. Die Vorgangsphase ändert sich nicht. Der Grund dafür ist, dass mit der Dauer der Vorgangsphasen die Gesamtausführungszeit unabhängig von den Betriebszuständen berechnet wird, die zum Maschinenstillstand geführt haben könnten. Es wird wieder der Betriebszustand *Produktion* erfasst.

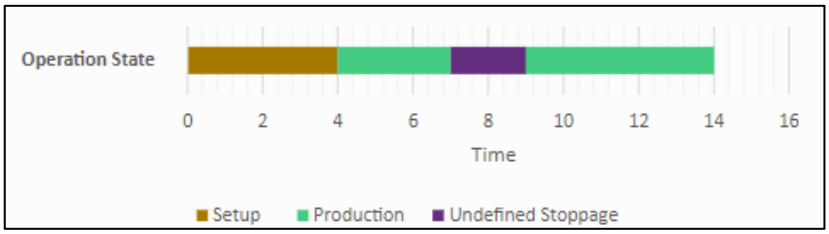

Bild 12: Erfasste Betriebszustände

Der Arbeiter muss am Shopfloor Terminal bestätigen, um festzustellen, was zu dem *unbegründeten Stillstand* geführt hat.

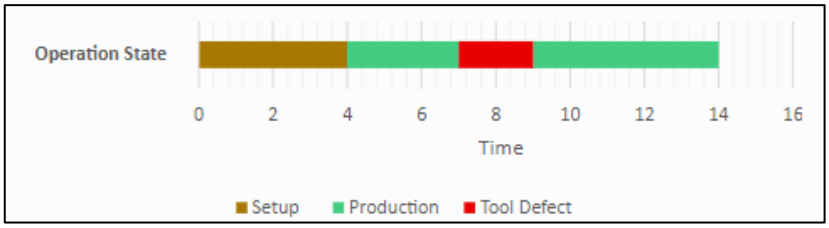

Bild 13: Erfasste Betriebszustände

2.5.1.4 Zeitbasen

Zeitbasen werden verwendet, um die Betriebszustände sinnvoll zu kategorisieren. Die folgenden Zeitbasen werden für die Berechnung der Kennzahlen benötigt.

2.5.1.4.1 Zeitbasis: Rüsten

Die Zeitbasis *Rüsten* enthält die Betriebszustände eines Arbeitsplatzes oder eines Vorgangs, die die Dauer der Rüstzeit beeinflussen.

Bei den Rüstzeiten ist zu beachten, dass diese generell vermieden werden sollen. Sie werden daher in interne und externe Rüstzeiten unterteilt.

- Interne Rüstzeiten bedeuten, dass die Arbeit gestoppt/angehalten werden muss, um z. B. ein Werkzeug zu wechseln.
- Externe Rüstzeiten bedeuten, dass die Bearbeitung am Arbeitsplatz nicht unterbrochen werden muss.

ⓘ Nur interne Rüstzeiten haben einen Einfluss auf die Verfügbarkeit und damit auf die OEE eines Arbeitsplatzes.

2.5.1.4.2 Zeitbasis: Produktion

Die Zeitbasis *Produktion* wird den Betriebszuständen eines Arbeitsplatzes bzw. Prozesses zugeordnet, die zur Dauer ihrer Produktionszeit beitragen.

2.5.1.4.3 Zeitbasis: geplante Schichtzeit

Die Zeitbasis *Geplante Schichtzeit* wird den Betriebszuständen eines Arbeitsplatzes bzw. Prozesses zugeordnet, die zur Dauer ihrer geplanten Schichtzeit beitragen.

Die geplante Schichtzeit umfasst die Schichten ohne Schichtpausen und ohne die geplanten Wartungszeiten. Arbeitsfreie Schichten (z. B. an Wochenenden, Feiertagen usw.) sind nicht enthalten.

Beispiel

Während der Bearbeitungsphase trat ein zusätzlicher Rüstvorgang eines geplanten Werkzeugwechsels auf. Dadurch war die erfasste Dauer des Betriebszustands *Rüsten* länger als die der Vorgangsphase *Einrichten*. Die Zeitbasen werden in der Abbildung visuell erläutert.

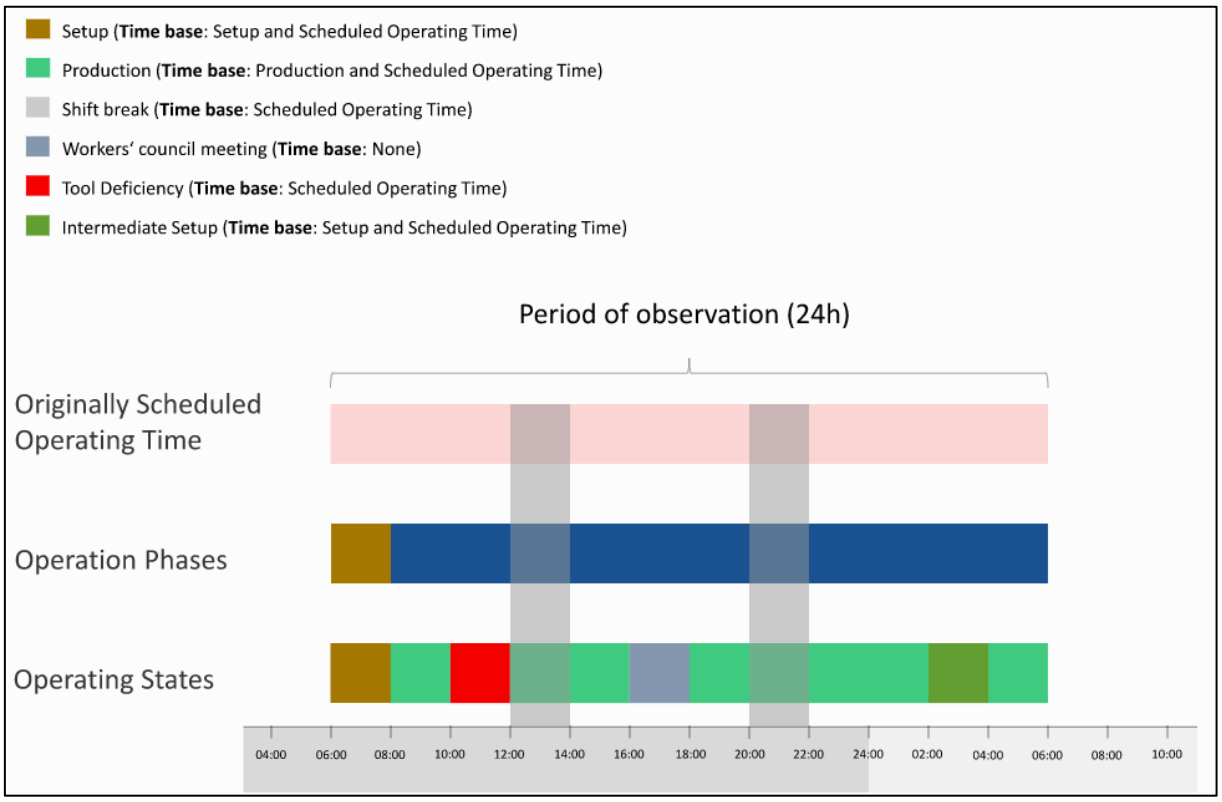

Bild 14: Übersicht der Zeitbasen

2.5.1.5 Beziehung zwischen Belegungszeit und Summe der Ausführungszeiten

Die Belegungszeit eines Arbeitsplatzes ist die Summe der Ausführungszeiten aller Vorgänge, während diese an dem eingeloggt angemeldet waren.

Beispiel:

Ein Vorgang begann an einem Arbeitsplatz um 6:00 Uhr morgens, wurde um 17:00 Uhr beendet und sofort an einem anderen Arbeitsplatz fortgesetzt.

Er wurde schließlich am folgenden Tag um 6:00 Uhr morgens beendet.

Dazu kommen zwei Schichtpausen von 12:00 bis 14:00 Uhr und von 20:00 bis 22:00 Uhr (insgesamt 4 Stunden).

Es gibt keine planmäßige Wartung.

Der *Beobachtungszeitraum* beträgt 24 Stunden. Daher errechnet sich die *geplante Schichtzeit* aus Beobachtungszeitraum minus *geplante Pausen* minus *geplante Wartung*.

$$geplante\ Betriebszeit = \begin{array}{l} Beobachtungszeitraum \\ -\ geplante\ Pausen \\ -\ geplante\ Wartung \end{array}$$

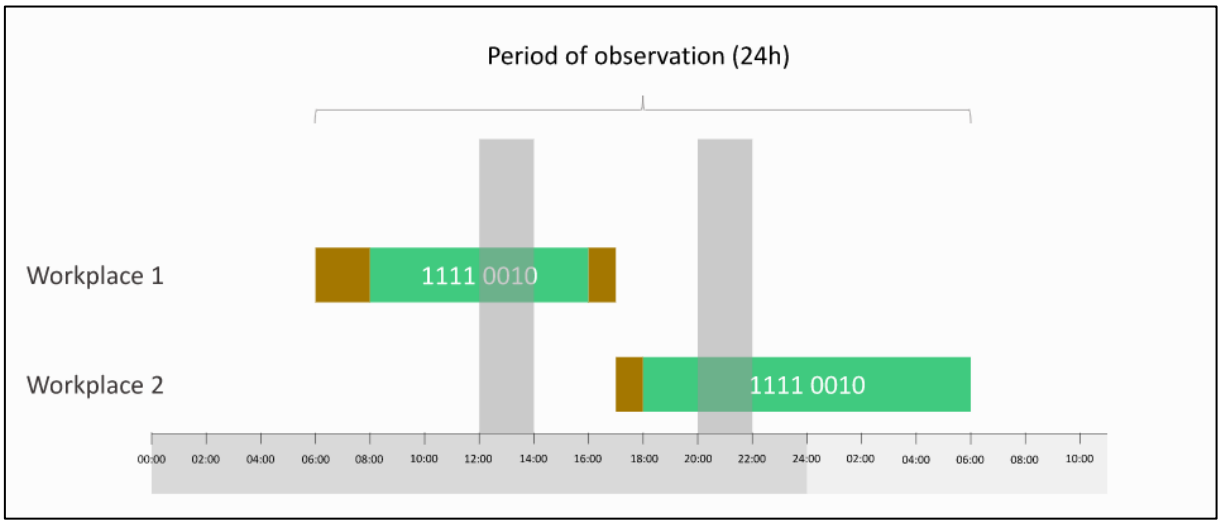

Bild 15: Beobachtungszeitraum (24h)

Daraus ergeben sich folgende *Betriebszeiten*:

Bearbeitungszeit	16 h
Rüstzeit	4 h
Ausführungszeit	20 h
Produktionszeit	16 h
Rüstphasenlänge	16 h

Daraus ergeben sich folgende Zeiten für den *Arbeitsplatz 1*:

geplante Betriebszeit	20 h
Bearbeitungszeit	6 h
Rüstzeit	3 h
Ausführungszeit	9 h
Produktionszeit	6 h

Daraus folgt:

$$Belegungsgrad = \frac{Belegungszeit}{geplante\ Betriebszeit} = \frac{9}{20} = 45\%$$

$$Rüstzeitverkürzung = \frac{Bearbeitungszeit}{Belegungszeit} = \frac{6}{9} = \frac{2}{3}$$

Daraus ergeben sich folgende Zeiten für den *Arbeitsplatz 2*:

geplante Betriebszeit	20 h
Bearbeitungszeit	10 h
Rüstzeit	1 h
Ausführungszeit	11 h
Produktionszeit	10 h

Daraus folgt:

$$Belegungsgrad = \frac{Belegungszeit}{geplante\ Betriebszeit} = \frac{11}{20} = 55\%$$

$$Setup\ reduction = \frac{Processing\ time}{Occupancy\ time} = \frac{10}{11}$$

2.5.2 Das digitale Abbild in FORCE Bridge API

2.5.2.1 Betriebszustände und Arbeitsplatzstatus

Betriebszustände werden definiert, um festzustellen, wie der aktuelle Zustand eines Arbeitsplatzes ist. Mögliche Betriebszustände eines Arbeitsplatzes sind z. B. *Produktion*, *Maschinenstörung*, *Werkzeugschaden*, etc.

Zu jedem Betriebszustand gehört ein Arbeitsplatzstatus, um festzustellen, ob ein Arbeitsplatz gerade arbeitet oder gestoppt ist. Sie sind wie folgt definiert.

Arbeitsplatzstatus	Beschreibung
DOWNTIME	Der Arbeitsplatz steht still.
PRODUCTION	Der Arbeitsplatz produziert.

ⓘ Die Betriebszustände können in der IoT-Plattform frei konfiguriert werden. Dadurch kann jeder Arbeitsplatz individuell angepasste Betriebszustände haben.

2.5.2.1.1 Vorgangsphasen

Zwischen der Freigabe eines Vorgangs und seiner Fertigstellung mit abschließender Rückmeldung an das ERP-System treten in der Produktion die nachfolgenden Phasen auf. Diese werden entweder am Shopfloor Terminal gemeldet oder von der IoT-Plattform automatisch aktualisiert:

Phase	Beschreibung
RELEASED	Der Produktionsauftrag eines Vorgangs wurde vom ERP-System freigegeben, aber der Vorgang wurde noch nicht gestartet oder im Detail geplant.
DISPATCHED	Der Vorgang wurde zu einem bestimmten Arbeitsplatz delegiert.
SETUP	Der Vorgang wurde an einem Arbeitsplatz angemeldet und dieser Arbeitsplatz ist für die Bearbeitung des Vorgangs gerüstet
PROCESSING	Der Vorgang wird bearbeitet.
INTERRUPTED	Der Vorgang wurde unterbrochen.
COMPLETED	Der Vorgang wurde vollständig abgeschlossen.
CLOSED	Der Vorgang wurde abgeschlossen, und eine abschließende Bestätigung wurde an das ERP-System gesendet.

2.5.2.2 Vorgangsbezogene Erfassung von Laufzeiten und Mengen (Betriebsdatenerfassung)

Ein Vorgang belegt einen Arbeitsplatz während der Rüstphase, der Bearbeitungsphase und der Abbauphase.

Sind keine Rüstmatrix und Fertigungsvarianten vorhanden, wird die Laufzeit dieser drei Phasen mit der Ressource *operations/{operationId}/specification* gefunden.

2.5.2.2.1 Werksdaten erfassen

Erfasste Zeitreihen

GET operations/{operationId}/recordedOperatingStates
Enthält alle Betriebszustände, die während dem Vorgang aufgetreten sind.

GET operations/{operationId}/recordedOperationPhases
Enthält alle Vorgangsphasen, die beim Vorgang aufgetreten sind.

GET operations/{operationId}/recordedOutputQuantities
Enthält alle Outputmengen (Gutmenge, Nacharbeit, Ausschuss), die bei dem Vorgang produziert wurden.

Verdichtete Werte

GET operations/{operationId}/recordedOperatingStates
GET operations/{operationId}/recordedOperationPhases

2.5.2.2.2 Maschinendatenerfassung

Istwerte

GET workplaces/{workplaceId}/recordedOperatingStates
Enthält alle Betriebszustände, die am Arbeitsplatz aufgetreten sind.

GET workplaces/{workplaceId}/recordedOperationPhases
Enthält alle Vorgangsphasen, die am Arbeitsplatz aufgetreten sind.

GET workplaces/{workplaceId}/recordedOutputQuantities
Enthält alle Outputmengen (Gutmenge, Nacharbeit, Ausschuss), die am Arbeitsplatz produziert wurden.

Verdichtete Werte

GET workplaces/{workplaceId}/recordedOperatingStates
GET workplaces/{workplaceId}/recordedOperationPhases

⚠ Jede Ressource enthält die Istwerte - oder für die Zusammenfassungen - die verdichteten Istwerte, die zur Berechnung der KPIs verwendet werden.

Video-Tutorial: FORCE Bridge API – Fertigungs-KPI

2.5.3 Tutorials

2.5.3.1 Soll- und Istwerte vergleichen

Die Soll-Rüstzeit eines Vorgangs sowohl mit der Rüstphasendauer aus der Ressource *operations/{operationId}/recordedOperationPhases* als auch mit der Rüstzeit aus *operations/{operationId}/operatingStateSummary* vergleichen. Ermitteln, ob der Sollwert der Rüstzeit eingehalten oder überschritten wurde.

Lösung:

Java

```
1.      IOperationClient operationClient = api.getOperationClient();
2.
3.      GetOperationsRequest operationsRequest = new GetOperationsRequest();
4.      operationsRequest
5.          .embed()
6.          .specification(true);
7.      Page<OperationResponse> operations = operationClient.getOperations(operationsRequest);
8.
9.      // Retrieve only one operation.
10.     OperationResponse operationResponse = operations
11.         .getResponse()
12.         .getElements()
13.         .get(0);
14.
15.     String operationId = operationResponse
16.         .getProperties()
17.         .getId();
18.
19.     OperationSpecificationPropertiesWSModel embedded = operationResponse
20.         .getSpecification()
21.         .getEmbedded();
22.     long targetSetupTime = embedded.getTargetSetupTime() + embedded.getTargetTeardownTime();
23.
24.     GetOperationOperatingStateSummaryRequest operatingStateSummaryRequest = new
    GetOperationOperatingStateSummaryRequest(operationId);
25.     OperationOperatingStateSummaryPage stateSummaryPage =
    operationClient.getOperatingStateSummary(operatingStateSummaryRequest);
26.     long actualSetupTime = stateSummaryPage
27.         .getResponse()
28.         .getProperties()
29.         .getSetupTime();
30.
31.     GetOperationRecordedOperationPhasesRequest phasesRequest = new
    GetOperationRecordedOperationPhasesRequest(operationId);
32.     OperationRecordedOperationPhasePage operationPhasePage =
    operationClient.getRecordedOperationPhases(phasesRequest);
33.
34.     OperationRecordedOperationPhasesResponse response = operationPhasePage.getResponse();
35.     long actualSetupPhaseDuration = response
36.         .getProperties()
37.         .getSetupPhaseDuration();
38.
39.     if (actualSetupTime <= targetSetupTime) {
40.         System.out.println("The setup time is within its target value.");
41.     } else {
42.         System.out.println("Setup time has been overstepped!");
43.     }
44.
45.     if (actualSetupPhaseDuration == targetSetupTime) {
46.         System.out.println("Target setup time and setup phase duration are identical!");
47.     } else {
48.         System.out.println("Target setup time and setup phase duration are NOT identical");
49.       }
```

/../_code/java/src/main/java/com/forcam/apps/ComparingTargetAndActualValues.java

2.5.3.2 Leistungskennzahlen (KPI) berechnen

2.5.3.2.1 Rüstzeit eines Arbeitsplatzes und eines Vorgangs berechnen

Die Rüstzeit wird aus der Summe aller Betriebszustände berechnet, die zur Rüstzeitbasis gehören. Die Summe wird für jeden Arbeitsplatz innerhalb des betrachteten Zeitraums berechnet.

Lösung:

Java

```
1.          //        final GetOperatingStateSummaryRequest stateSummaryRequest = new
    GetOperatingStateSummaryRequest(mWorkplaceUUID); //FIXME inconsistent naming.
2.          //        final WorkplaceOperatingStateSummaryResponse stateSummaryResponse =
    mWorkplaceClient.getOperatingStateSummary(stateSummaryRequest);
3.          //        final long summarySetupTime = stateSummaryResponse
4.          //            .getProperties()
5.          //            .getSetupTime();
6.          //
7.          //        final GetWorkplaceRecordedOperatingStatesRequest
    operatingStatesRequest = new
    GetWorkplaceRecordedOperatingStatesRequest(mWorkplaceUUID);
8.          //        final RecordedOperatingStateCollectionResponse
    stateCollectionResponse =
    mWorkplaceClient.getRecordedOperatingStates(operatingStatesRequest);
9.          //
10.         //        long recordedSetupTime = 0L;
11.         //        for (OperatingStateRecordWSModel record : stateCollectionResponse
12.         //            .getProperties()
13.         //            .getElements()) {
14.         //            long duration = record
15.         //                .getTimePeriod()
16.         //                .getDuration(); //FIXME duration is not properly calculated
    because, if the setup is still in progress the end date does not get set!
17.         //            recordedSetupTime += duration;
18.         //        }
```

/../_code/java/src/test/java/com/forcam/durations/workplace/SetupTimeTestExecutable.java

Die Rüstzeit kann auch für einen Vorgang berechnet werden.

ⓘ Wenn der Vorgang an mehreren Arbeitsplätzen gerüstet und bearbeitet wurde, müssen die Rüstzeiten aller Arbeitsplätze, an denen der Vorgang ausgeführt wurde, addiert werden.

Lösung:

Java

```
1.      //        final GetOperationOperatingStateSummaryRequest stateSummaryRequest =
    new GetOperationOperatingStateSummaryRequest(mOperationUUID);
2.      //        final OperationOperatingStateSummaryResponse stateSummaryResponse =
    mOperationClient.getOperatingStateSummary(stateSummaryRequest);
3.      //        final long summarySetupTime = stateSummaryResponse
4.      //            .getProperties()
5.      //            .getSetupTime();
6.      //
7.      //        final GetOperationRecordedOperatingStatesRequest operatingStatesRequest
    = new GetOperationRecordedOperatingStatesRequest(mOperationUUID);
8.      //        final RecordedOperatingStateCollectionResponse stateCollectionResponse
    = mOperationClient.getRecordedOperatingStates(operatingStatesRequest);
9.      //
10.     //        final long recordedSetupTime = stateCollectionResponse
11.     //            .getProperties()
12.     //            .getElements()
13.     //            .stream()
14.     //            .mapToLong(record -> record
15.     //                .getTimePeriod()
16.     //                .getDuration())
17.     //            .sum();
```

/../_code/java/src/test/java/com/forcam/durations/operation/SetupTimeTestExecutable.java

2.5.3.2.2 Die Ausführungszeit eines Vorgangs berechnen

Die Ausführungszeit ist die Summe der Laufzeiten aller für den Vorgang anfallenden Rüstphasen und Bearbeitungsphasen.

Lösung:

Java

```
//        final GetOperationRecordedOperationPhasesRequest operationPhasesRequest = new GetOperationRecordedOperationPhasesRequest(mOperationUUID);
//        final OperationRecordedOperationPhasesResponse recordedOperationPhasesResponse = mOperationClient.getRecordedOperationPhases(operationPhasesRequest);
//
//        final long executionTime = recordedOperationPhasesResponse
//            .getProperties()
//            .getExecutionTime();
//
//        final long setupPhaseDuration = recordedOperationPhasesResponse
//            .getProperties()
//            .getSetupPhaseDuration();
//        final long processingPhaseDuration = recordedOperationPhasesResponse
//            .getProperties()
//            .getProcessingPhaseDuration();
//
//        final long actualExecutionTime = processingPhaseDuration + setupPhaseDuration;
```

/../_code/java/src/test/java/com/forcam/durations/operation/ExecutionTimeTestExecutable.java

2.5.3.2.3 Die Bearbeitungszeit eines Vorgangs berechnen

Die Bearbeitungszeit eines Vorgangs ist die Ausführungszeit abzüglich der Rüstzeiten.

Lösung:

Java

```
//        final GetOperationOperatingStateSummaryRequest stateSummaryRequest = new GetOperationOperatingStateSummaryRequest(mOperationUUID);
//        final OperationOperatingStateSummaryResponse stateSummaryResponse = mOperationClient.getOperatingStateSummary(stateSummaryRequest);
//        final OperatingStateSummaryPropertiesWSModel properties = stateSummaryResponse.getProperties();
//
//        final long summaryProcessingTime = properties.getProcessingTime();
//
//        final long executionTime = properties.getExecutionTime();
//        final long setupTime = properties.getSetupTime();
//
//        final long actualProcessingTime = executionTime - setupTime;
```

/../_code/java/src/test/java/com/forcam/durations/operation/ProcessingTimeTestExecutable.java

2.5.3.2.4 Die Belegungszeit eines Arbeitsplatzs berechnen

Die Belegungszeit eines Arbeitsplatzes wird aus der Summe aller Ausführungszeiten aller Vorgänge berechnet, die an diesem Arbeitsplatz innerhalb des betreffenden Zeitraums ausgeführt wurden.

⚠ Durchführen mit *workplaces/{workplaceId}/recordedOperationPhases*

Lösung:

Java

```
1.          //        final GetWorkplaceRecordedOperatingStatesRequest operatingStatesRequest = new
    GetWorkplaceRecordedOperatingStatesRequest(mWorkplaceUUID);
2.          //        final RecordedOperatingStateCollectionResponse operatingStateCollectionResponse =
    mWorkplaceClient.getRecordedOperatingStates(operatingStatesRequest);
3.          //
4.          //        final WorkplaceOperatingStateSummaryResponse operatingStateSummaryResponse =
    mWorkplaceClient.getOperatingStateSummary(new GetOperatingStateSummaryRequest(
5.          //            mWorkplaceUUID));
6.          //
7.          //        final Long expectedOccupancyTime = operatingStateSummaryResponse
8.          //            .getProperties()
9.          //            .getOccupancyTime();
10.         //
11.         //        //TODO: get all operations that have been executed at this workplace within a
    specified time period.
12.         //        //FIXME: the above mentioned action cannot be easily achieved, because there is
    no such interface.
13.         //        fail(); //TODO remove me once implemented.
14.         //
15.         //        final Long actualOccupancyTime = 0L;
```

/../_code/java/src/test/java/com/forcam/durations/workplace/OccupancyTimeTestExecutable.java

2.5.3.2.5 Vergleich zwischen ursprünglich geplante Betriebszeiten mit den Zeiten die später als geplante Betriebszeiten angegeben wurden

ⓘ Die Dauer der geplanten Betriebszeit kann von der ursprünglich geplanten Betriebszeit abweichen. Ein Beispiel: Es fand eine ungeplante Besprechung statt, die keine Auswirkungen auf die Verfügbarkeit der Arbeitsplätze haben soll und deshalb nicht der Zeitbasis "Geplante Betriebszeit" zugeordnet wird.

Lösung:

Java

```
1.          final WorkplaceOperatingStateSummaryPage stateSummaryResponse =
    mWorkplaceClient.getOperatingStateSummary(new GetOperatingStateSummaryRequest(
2.              mWorkplaceUUID));
3.          final Long scheduledOperatingTime = stateSummaryResponse
4.              .getResponse()
5.              .getProperties()
6.              .getScheduledOperatingTime();
7.
8.          final WorkplaceScheduledOperatingTimesCollectionPage
    operatingTimesCollectionResponse = mWorkplaceClient.getScheduledOperatingTimes(new
    GetWorkplaceScheduledTimesRequest(
9.              mWorkplaceUUID));
10.         //FIXME throws 'Unexpected internal server error'
11.         long originallyScheduledOperatingTime = 0L;
12.         for (TimePeriodWSModel timePeriodWSModel : operatingTimesCollectionResponse
13.             .getResponse()
14.             .getProperties()
15.             .getElements()) {
16.             long duration = timePeriodWSModel.getDuration();
17.             originallyScheduledOperatingTime += duration;
18.         }
```

/../_code/java/src/test/java/com/forcam/durations/workplace/ScheduledOperatingTimeTestExecutable.java

2.5.3.3 Leistungskennzahlen (KPIs) abfragen

FORCE Bridge enthält bereits alle erforderlichen KPIs für die OEE eines Vorgangs. Diese KPIs können wie in den nachfolgenden Beispielen abgerufen werden:

Lösung für Vorgang:

Java

```
1.              Page<OperationResponse> operationResponsePage =
    operationClient.getOperations(new GetOperationsRequest());
2.
3.              operationResponsePage
4.                  .streamForward()
5.                  .map(OperationResponse::getProperties)
6.                  .map(OperationPropertiesWSModel::getId)
7.                  .forEach(operationId -> {
8.
9.                      GetOperationOperatingStateSummaryRequest
    operatingStateSummaryRequest = new
    GetOperationOperatingStateSummaryRequest(operationId);
10.                     GetOperationQuantitySummaryRequest quantitySummaryRequest = new
    GetOperationQuantitySummaryRequest(operationId);
11.
12.                     // Retrieving the operating state summary and quantity summary of
    an operation
13.                     try {
14.                         OperationOperatingStateSummaryPage
    operatingStateSummaryResponse =
    operationClient.getOperatingStateSummary(operatingStateSummaryRequest);
15.                         OperatingQuantitySummaryCollectionPage
    quantitySummaryCollectionResponse =
    operationClient.getQuantitySummary(quantitySummaryRequest);
16.
17.                         final double availability = operatingStateSummaryResponse
18.                             .getResponse()
19.                             .getProperties()
20.                             .getProcessAvailability();
21.
22.                         final double performance = quantitySummaryCollectionResponse
23.                             .getResponse()
24.                             .getProperties()
25.                             .getPerformance();
26.
27.                         final double qualityRate = quantitySummaryCollectionResponse
28.                             .getResponse()
29.                             .getProperties()
30.                             .getQualityRate();
31.
32.                         double overalEquipmentEfficency = availability * performance *
    qualityRate;
33.
34.                     } catch (ForceAPIException e) {
35.                         e.printStackTrace();
36.                     }
37.                 });
```

/../_code/java/src/main/java/com/forcam/apps/OperationSummaries.java

Lösung für Arbeitsplatz:

Java

```
38.             final Page<WorkplaceResponse> workplaceResponsePage =
    workplaceClient.getWorkplaces(new GetWorkplacesRequest());
1.
2.              workplaceResponsePage
3.                  .streamForward()
4.                  .map(WorkplaceResponse::getProperties)
5.                  .map(WorkplacePropertiesWSModel::getId)
6.                  .forEach(workplaceId -> {
7.                      final GetOperatingStateSummaryRequest operatingStateSummaryRequest
    = new GetOperatingStateSummaryRequest(workplaceId);
8.                      final GetWorkplaceQuantitySummaryRequest
    workplaceQuantitySummaryRequest = new GetWorkplaceQuantitySummaryRequest(workplaceId);
9.
10.                     // Retrieving the operating state summary and quantity summary of
    a workplace
11.                     try {
12.                         final WorkplaceOperatingStateSummaryPage
    operatingStateSummaryResponse = workplaceClient.getOperatingStateSummary(
13.                             operatingStateSummaryRequest);
14.
15.                         final WorkplaceQuantitySummaryCollectionPage
    quantitySummaryCollectionResponse = workplaceClient.getQuantitySummary(
16.                             workplaceQuantitySummaryRequest);
17.                     } catch (ForceAPIException e) {
18.                         LOGGER.error(e.getMessage());
19.                     }
20.                 });
```

/../_code/java/src/main/java/com/forcam/apps/WorkplaceSummaries.java

Overall equipment efficiency (OEE) berechnen

⚠ Das Beispiel der OEE Berechnung sowohl für *Vorgang* als auch für *Arbeitsplatz* verwenden.

3 FORCE Bridge API als Integrationsplattform

Hier wird erklärt, wie FORCE Bridge API die organisatorische Interoperabilität aller Anwendungen, IT-Systeme und Personen in einer Fertigungsumgebung sicherstellt. Es werden typische Anwendungen als Beispiele für die Anwendungsentwicklung erläutert. Außerdem stehen für die jeweiligen Anwendungen Templates für die Interaktion mit der IoT-Plattform zur Verfügung.

3.1 Applications for manufacturing operations management (MOM)

The essential principle of an Industrial IoT platform is the strict separation of acquisition systems and the applications operating on their data. While machine data collection and production data collection are carried out in the platform itself, all applications are supplied with the acquired data via the application programming interface.

Industrial IoT Plattformdienste	Eingabe- oder Ausgabedaten
Maschinendatenerfassung	Teilressourcen von Arbeitsplätzen
Bestellungsdatenerfassung	Teilressourcen des Vorgangs
Prozessdatenerfassung	Teilressourcen der Geräte
Energieverbrauchsdatenerfassung	Teilressourcen der Geräte
Erfassung der Arbeitsaktivität	Teilressourcen der Mitarbeiter
Track & Trace	Teilressourcen der Vorgänge und Geräte
Distributed-Numerical-Control (DNC)	Dokumente

FORCE Bridge API unterstützt alle verfügbaren Anwendungen für die Fertigungsverwaltung. Dies sind nicht nur Anwendungen, die bisher Teil eines MES waren, sondern auch die neuesten und zukünftigen Technologien für Smart Manufacturing. Folgende Tabelle gibt einen beispielhaften Überblick über einige typische Anwendungen.

Applikation	Beschreibung
Shopfloor Visualisierung	Onlinevisualisierung der Maschinen, Handarbeitsplätze und des Produktionsfortschritts
Leistungsanalyse	Reports, Dashboards, KPIs
Alarmierung	Alarmierung über Ereignisse oder unerwünschte Zustände auf mobilen Apps, per SMS, oder E-Mail
Digitale Plantafel	Digitale Plantafel für detaillierte Auftragsplanung oder Arbeitskapazitätsplanung
Dynamische Auftragsplanung	Die Feinplanung von Arbeitsplätzen im kurzfristigen zeitlichen Bereich, die unter Berücksichtigung aller aktuell verfügbaren

Applikation	Beschreibung
	Informationen kontinuierlich das optimale Planungsszenario ermittelt.
Aktive Fertigungssteuerung	Fertigungssteuerung, die auf Basis der ursprünglichen Auftragsfeinplanung, und unter Berücksichtigung aller aktuell verfügbaren Informationen, kontinuierlich eine Prognose für die tatsächliche Auftragsabwicklung erstellt. Beim Auftreten von kritischen Verzögerungen oder unvorhergesehenen Ereignissen werden vorbeugende Maßnahmen vorgeschlagen.
Dynamische Arbeitskapazitätsplanung	Arbeitskapazitätsplanung, die laufend das optimale Planungsszenario, unter Berücksichtigung aller aktuell verfügbaren Informationen, ermittelt.
Applikationen zur Optimierung von planungsrelevanten Vorgabewerten oder Voraussetzungen, um die Planungsstabilität zu verbessern.	Applikationen aus dem Bereich des maschinellen Lernens, die durch Analyse der aufgezeichneten historischen Daten Prognosen für Rüst- und Bearbeitungszeiten liefern.
Vorausschauende Wartung	Applikationen aus dem Bereich des maschinellen Lernens, die durch Analyse der aufgezeichneten Prozessdaten Vorhersagen über drohende Fehlfunktionen liefern.
Aktive Qualitätssicherung	Applications from the field of machine learning that warn of imminent quality losses by analyzing the recorded process data.
Fertigungsdatenmanagement	Verwaltung von Fertigungsdaten und deren Synchronisation mit den Product-Lifecycle-Management-Systemen.

Die FORCE Bridge API gibt Softwareentwicklern die Möglichkeit, Applikationen zu entwickeln ohne Kenntnis über komplizierte domänenspezifische Zusammenhänge und völlig unabhängig von anderen Applikationen. Das Ressourcendesign der API sorgt dafür, dass keine Applikation Kenntnisse über eine andere Applikation haben muss. Komplexe Ereignisse informieren Applikationen, wenn die von ihnen bereitgestellten Informationen aktualisiert werden müssen.

3.2 Callbacks und Ereignisse

Mit der Bridge-API können Applikationen und Drittsysteme über Ereignisse via HTTP oder MQTT benachrichtigt werden. Dafür muss man sich registrieren. Dazu wird ein Callback per POST erstellt, der die Art des Ereignisses angibt, über das die Applikation oder das Drittsystem benachrichtigt werden soll, und die URL, unter der es erreicht werden kann. Für die Benachrichtigung über MQTT muss ein Topic definiert werden, das immer mit *external* beginnen muss, um von den intern in der IoT-Plattform verwendeten Topics zu unterscheiden:

"url": "mqtt://{host}:1883/external/{any topic}"

Drittsysteme, die nicht über MQTT, sondern über HTTP über Ereignisse informiert werden sollen, benötigen einen eigenen HTTP-Server. In diesem Fall ist das Callback selbst ein HTTP-POST-Request mit JSON-Daten, der wie folgt an die URL des zu spezifizierenden HTTP-Servers gesendet wird:

"url": **Fehler! Linkreferenz ungültig.**

3.2.1 Callbacks

3.2.1.1 Rückrufanfrage an den externen Webservice

Immer, wenn ein Ereignis eintritt, das mit einer bestehenden Registrierung übereinstimmt, werden die Ereignisdaten über eine HTTP-POST-Anfrage an die angegebene URL gesendet.

3.2.1.2 Wiederholungen

Wenn ein Callback fehlschlägt (z. B. wenn das System, das den registrierten Endpunkt bereitstellt, ausfällt), werden alle 10 Sekunden Wiederholungsversuche ausgelöst, bis die maximale Anzahl von Wiederholungsversuchen erreicht ist. Wenn die maximale Anzahl von Wiederholungsversuchen erreicht ist, wird das Callback verworfen und es werden keine weiteren Wiederholungsversuche für dieses Callback gesendet. Wiederholungen bedeuten, dass Callbacks möglicherweise nicht in der richtigen Reihenfolge gesendet werden. Daher muss der Zeitstempel des Ereignisses, das den Callback ausgelöst hat, vom empfangenden System berücksichtigt werden. Wenn die maximale Anzahl der fehlgeschlagenen Callbacks erreicht ist, wird das älteste verworfen, auch wenn die maximale Anzahl der Wiederholungsversuche noch nicht erreicht ist.

3.2.1.3 Ein Callback erstellen

Um über Ereignisse benachrichtigt zu werden, muss ein Callback erstellt werden. Dies geschieht durch Aufrufen von POST-Callbacks mit dem folgenden Inhalt im Anfragetext:

```
1.  {
2.      "objectFilter": [
3.          {
4.          "name": "string",
5.          "value": "string"
6.          }
7.      ],
8.      "eventType": "COMMAND",
9.      "eventName": "string",
10.     "url": "string",
11.     "maxRedeliverAttempts": 0,
12.     "maxUnconfirmedMessages": 0
13. }
```

Es müssen mindestens zwei Angaben in der Anfrage gemacht werden: Der Ereignistyp, über den informiert werden soll, und die URL, an der die Information über ein Ereignis gesendet werden soll.

Ereignisse, die sich auf einen bestimmten Typ von Primärressourcen beziehen, können bei der Erstellung des Callbacks auf eine Teilmenge oder eine einzelne Primärressource eingeschränkt werden. Dazu werden Objektfilter verwendet, die als eine Feldanordnung von Namen/Werte-Paaren übergeben werden. *Name* gibt eine Eigenschaft des Referenzobjekts an. *Wert* ist ein Wert, den die Eigenschaft haben muss, um das Filterkriterium anwenden zu können. In der aktuellen Version 2 wird nur die Eigenschaft *ID* unterstützt, d. h. die Referenzobjekte eines Ereignistyps können nur nach UUIDs gefiltert werden.

Alternativ kann den vom Callback angegebenen Ereignissen ein Ereignisname zugeordnet werden.

Java Beispiel mit http

```
1.          final ICallbackClient callbackClient = api.getCallbackClient();
2.
3.          callbackClient.addListener(new AbstractOperationPhaseCommandListener() {
4.              @Override
5.              public void handle(EventModel<OperationPhaseCommandDataWsModel>
    eventModel) {
6.                  OperationPhaseCommandDataWsModel data = eventModel.getData();
7.                  /* process received data */
8.              }
9.
10.         });
11.
12.         try {
13.             callbackClient.startCallbackHttpServer(8081);
14.         } catch (IOException e) {
15.             LOGGER.error(e.getMessage());
16.         }
17.
18.         try {
19.             CreateCallbackRequest callbackRequest = new CreateCallbackRequest();
20.             callbackRequest.setEvent(Event.COMMAND__OPERATION_PHASE);
21.             callbackRequest.setMaxRedeliveryAttempts(5);
22.             callbackRequest.setMaxUnconfirmedMessages(5);
23.             callbackRequest.setUrl("http://localhost:8081/callbacks");
24.             callbackClient.createCallback(callbackRequest);
25.         } catch (ForceAPIException e) {
26.             LOGGER.error(e.getMessage());
27.         }
```

Download

Java Beispiel mit MQTT

```
1.          String callbackId = null;
2.
3.          final ICallbackClient callbackClient = api.getCallbackClient();
4.          callbackClient.addListener(new AbstractOperationPhaseCommandListener() {
5.
6.              @Override
7.              public void handle(EventModel<OperationPhaseCommandDataWsModel> eventModel) {
8.                  OperationPhaseCommandDataWsModel data = eventModel.getData();
9.                  /* process received data */
10.                 String operationId = data.getOperationId();
11.                 OperationPhase operationPhaseId = data.getOperationPhaseId();
12.                 switch (operationPhaseId) {
13.                     case RELEASED:
14.                     case DISPATCHED:
15.                     case SETUP:
16.                     case TRAINING:
17.                     case PROCESSING:
18.                     case INTERRUPTED:
19.                     case COMPLETED:
20.                     case CLOSED:
21.                         LOGGER.info("Operation [{}] is in phase {}", operationId,
    operationPhaseId);
22.                         break;
23.                     default:
24.                         LOGGER.warn("Unknown Operation Phase!");
25.                 }
26.
27.             }
28.
29.             @Override
30.             public void connectionLost(CallbackProvider callbackProvider) {
31.                 /* The connection lost event can only be captured with the stateful mqtt, remember
    http is stateless! */
32.                 switch (callbackProvider) {
33.                     case MQTT:
34.
35.                         break;
36.                     default:
37.                         LOGGER.warn("Unknown Callback Provider!");
38.                 }
39.             }
40.         });
41.
42.         try {
```

```
            MqttConfiguration config = new MqttConfiguration("tcp://localhost:1883");
            callbackClient.connectToMqttBroker(config);
        } catch (MqttException e) {
            LOGGER.error(e.getMessage());
        }

        CreateCallbackRequest callbackRequest = new CreateCallbackRequest();
        callbackRequest.setEvent(Event.COMMAND__OPERATION_PHASE);
        final CallbackObjectFilterWSModel objectFilter = new CallbackObjectFilterWSModel();
        objectFilter.setName("id");
        objectFilter.setValue("DA892519507444768080F0E81879513");
        request.setObjectFilter(Collections.singletonList(objectFilter));
        callbackRequest.setUrl("mqtt://third-party-system:1883/external/OperationPhaseChanged");

        CallbackResponse callbackResponse = null;
        try {
            callbackResponse = callbackClient.createCallback(callbackRequest);
        } catch (ForceAPIException e) {
            LOGGER.error(e.getMessage());
        }

        assert callbackResponse != null;

        callbackId = callbackResponse
            .getProperties()
            .getId();
```

Download

CURL Beispiel mit CURL

```
curl -X POST http://$HOST:$PORT/ffwebservice/api/v2/callbacks \
  --header "Content-Type: application/json" \
  --header "Accept: application/json" \
  --header "Authorization: Bearer $TOKEN" \
  -d '{
     "objectFilter": [
       {
         "name": "id",
         "value": "DA892519507444768080F0E81879513”"
       }
     ],
     "eventName": "Operation 100 has changed phase",
     "eventType": "OPERATION_PHASE_CHANGED",
     "url": “http://third-party-system:24080/webserver”
  }
```

CURL Beispiel mit MQTT

```
curl -X POST http://$HOST:$PORT/ffwebservice/api/v2/callbacks \
  --header "Content-Type: application/json" \
  --header "Accept: application/json" \
  --header "Authorization: Bearer $TOKEN" \
  -d '{
     "objectFilter": [
       {
         "name": "id",
         "value": "DA892519507444768080F0E81879513"
       }
     ],
     "eventName": "Operation 100 has changed phase",
     "eventType": "OPERATION_PHASE_CHANGED",
     "url": "mqtt://third-party-system:1883/external/OperationPhaseChanged"
  }
```

3.2.1.4 Alle registrierten Callbacks abrufen

Um alle registrierten Callbacks zu erhalten, muss eine Anfrage an **callbacks/ via GET** gesendet werden.

Beispiel JAVA

```
1.          CollectionRequest getCallBackRequest = new CollectionRequest();
2.          //FIXME Apparently there is no GetCallbackRequest Object. CollectionRequest is
    to generic to set any 'body'!
3.          Page<CallbackResponse> callbackResponsePage =
    callbackClient.getCallbacks(getCallBackRequest);
```

Beispiel CURL

```
1.  curl -X GET "http://$HOST:$PORT/ffwebservice/api/v1/callbacks/$CALLBACK_UUID" \
2.    --header "Content-Type: application/json" \
3.    --header "Accept: application/json" \
```

3.2.1.5 Einen registrierten Callback löschen

Um einen registrierten Callback zu löschen, muss eine Anfrage über **callbacks/{callbackId} via DELETE** gesendet werden, wobei *callbackId* die ID des Callbacks ist.

Beispiel JAVA

```
1.          DeleteCallbackRequest deleteCallbackRequest = new
    DeleteCallbackRequest(callbackUUID);
2.          callbackClient.deleteCallback(deleteCallbackRequest);
```

Beispiel CURL

```
1.  curl -X DELETE "http://$HOST:$PORT/ffwebservice/api/v1/callbacks/$CALLBACK_UUID" \
2.    --header "Content-Type: application/json" \
3.    --header "Accept: application/json" \
```

3.2.2 Ereignistypen

Eine Liste aller verfügbaren Rückrufe werden im Swagger unter **/ffwebservices/docs/index.html#callback-api** , und Befehle unter **/ffwebservices/docs/index.html#command-api** gefunden.

Im Swagger sind alle Ereignistypen aufgelistet, für die eine Callback-Registrierung möglich ist. Die darin aufgeführten Ereignisse sind in der Regel die einzigen, die von Applikationen benötigt werden. Sie werden als verdichtete Informationen von der IoT-Plattform als Folge bestimmter anderer Einzelereignisse bereitgestellt. Darüber hinaus werden zahlreiche Ereignisse generiert, die über Details informieren.

Die Ereignistypen *COMMAND*, *SHOP_FLOOR_TERMINAL* und *INTERNAL* sind für interne Ereignisse der IoT-Plattform reserviert.

3.2.3 Ereignisse empfangen

Tritt ein Ereignis ein (in diesem Fall ein *OPERATION_PHASE _CHANGED* Ereignis), für das sich ein Drittsystem per HTTP-Callback registriert hat, erhält der HTTP-Server des Drittsystems beispielsweise die folgende POST-Anfrage:

```
<curl -X POST
  --header 'Content-Type: application/json'
  -d '{
  "properties" : {
    "callbackId" : "B1B1BFB60ECF4DB7B095F096992FB9FA",
    "timestamp" : "2019-03-06T11:04:16.786Z",
    "data" : {
      "currentOperationPhase" : "PROCESSING",
      "previousOperationPhase" : "INTERRUPTED",
      "operationId" : "2A05AAEF15FD41A8A4A36C45516871AA"
    },
    "objectId" : "DA8925195074447680 80F0E81879513",
    "objectType" : "OPERATION",
    "eventType" : "OPERATION_PHASE_CHANGED",
    "eventName" : "Operation 100 has changed phase"
  },
  "_links" : {
    "callback" : {
      "method" : "GET",
      "embeddable" : true,
      "href" : "http://localhost:24080/ffwebservices/api/v2/callbacks/B1B1BFB60ECF4DB7B095F096992FB9FA"
    }
  }
}' 'CALLBACK_URL'>
```

Es ist möglich, eine Basic-Auth-Authentifizierung für Rückrufe zu konfigurieren. Wenn ein Ereignis eintritt, für das sich eine Applikation über MQTT registriert hat, erhält die Applikation eine MQTT-Nachricht.

Beispiel in Java:

```
4.  ////////////////////////////////////////////////////////////////////////////////
5.  //
6.  // Title:   Simple detailed order planning
7.  // Created: 24.09.2018
8.  // Author:  David Zeise
9.  // Source:  https://forcebridge.io
10. // License: CC BY 3.0 (https://creativecommons.org/licenses/by/3.0/)
11. //
12. ////////////////////////////////////////////////////////////////////////////////
13.
14. package com.forcam.usage.callback;
15.
16. import com.forcam.common.extension.BridgeAPIProvider;
17. import com.forcam.na.ffwebservices.client.BridgeAPI;
18. import com.forcam.na.ffwebservices.client.api.callback.CallbackProvider;
19. import com.forcam.na.ffwebservices.client.api.callback.EventModel;
20. import com.forcam.na.ffwebservices.client.api.callback.ICallbackClient;
21. import com.forcam.na.ffwebservices.client.api.callback.MqttConfiguration;
22. import
    com.forcam.na.ffwebservices.client.api.callback.listener.AbstractOperationPhaseCommandListener;
23. import com.forcam.na.ffwebservices.client.api.callback.request.CreateCallbackRequest;
24. import com.forcam.na.ffwebservices.client.api.exception.ForceAPIException;
25. import com.forcam.na.ffwebservices.model.callback.CallbackResponse;
26. import com.forcam.na.ffwebservices.model.command.operation.OperationPhaseCommandDataWsModel;
27. import com.forcam.na.ffwebservices.model.definition.OperationPhase;
28. import com.forcam.na.ffwebservices.model.event.Event;
29. import org.eclipse.paho.client.mqttv3.MqttException;
30. import org.junit.jupiter.api.*;
31. import org.junit.jupiter.api.extension.*;
32. import org.slf4j.Logger;
33. import org.slf4j.LoggerFactory;
34.
35. /**
36.  *
37.  */
38. @ExtendWith(BridgeAPIProvider.class)
39. class CallbackCreateMQTTTest {
40.
41.     // ---------------------------------------------------------------------
42.     // constants
43.     // ---------------------------------------------------------------------
```

```
44.
45.     private static final Logger LOGGER = LoggerFactory.getLogger(CallbackCreateMQTTTest.class);
46.
47.     // ------------------------------------------------------------------------
48.     // tests
49.     // ------------------------------------------------------------------------
50.     @Test
51.     void test(BridgeAPI api) {
52.         //<editor-fold example="1" desc="Create a callback, that uses MQTT">
53.
54.         String callbackId = null;
55.
56.         final ICallbackClient callbackClient = api.getCallbackClient();
57.         callbackClient.addListener(new AbstractOperationPhaseCommandListener() {
58.
59.             @Override
60.             public void handle(EventModel<OperationPhaseCommandDataWsModel> eventModel) {
61.                 OperationPhaseCommandDataWsModel data = eventModel.getData();
62.                 /* process received data */
63.                 String operationId = data.getOperationId();
64.                 OperationPhase operationPhaseId = data.getOperationPhaseId();
65.                 switch (operationPhaseId) {
66.                     case RELEASED:
67.                     case DISPATCHED:
68.                     case SETUP:
69.                     case TRAINING:
70.                     case PROCESSING:
71.                     case INTERRUPTED:
72.                     case COMPLETED:
73.                     case CLOSED:
74.                         LOGGER.info("Operation [{}] is in phase {}", operationId,
    operationPhaseId);
75.                         break;
76.                     default:
77.                         LOGGER.warn("Unknown Operation Phase!");
78.                 }
79.
80.             }
81.
82.             @Override
83.             public void connectionLost(CallbackProvider callbackProvider) {
84.                 /* The connection lost event can only be captured with the stateful mqtt, remember
    http is stateless! */
85.                 switch (callbackProvider) {
86.                     case MQTT:
87.
88.                         break;
89.                     default:
90.                         LOGGER.warn("Unknown Callback Provider!");
91.                 }
92.             }
93.         });
94.
95.         try {
96.             MqttConfiguration config = new MqttConfiguration("tcp://localhost:1883");
97.             callbackClient.connectToMqttBroker(config);
98.         } catch (MqttException e) {
99.             LOGGER.error(e.getMessage());
100.        }
101.
102.        CreateCallbackRequest callbackRequest = new CreateCallbackRequest();
103.        callbackRequest.setEvent(Event.COMMAND__OPERATION_PHASE);
104.        final CallbackObjectFilterWSModel objectFilter = new CallbackObjectFilterWSModel();
105.        objectFilter.setName("id");
106.        objectFilter.setValue("DA8925195074447680 80F0E81879513");
107.        request.setObjectFilter(Collections.singletonList(objectFilter));
108.        callbackRequest.setUrl("mqtt://third-party-system:1883/external/OperationPhaseChanged");
109.
110.        CallbackResponse callbackResponse = null;
111.        try {
112.            callbackResponse = callbackClient.createCallback(callbackRequest);
113.        } catch (ForceAPIException e) {
114.            LOGGER.error(e.getMessage());
115.        }
116.
117.        assert callbackResponse != null;
118.
119.        callbackId = callbackResponse
120.            .getProperties()
121.            .getId();
122.
123.        //</editor-fold>
124.    }
125.
126.}
```

3.2.4 Selbst definierte Ereignisse an Applikationen oder Drittsysteme senden

Eine Applikation kann Ereignisse an andere Applikationen oder Drittsysteme senden. Dazu müssen POST-Ereignisse mit dem folgenden Inhalt im Anfragetext aufgerufen werden:

```
{
"data": {},
"eventName": "string",
"eventType": "EXTERNAL",
"objectId": "string",
"objectType": "OPERATION",
"timestamp": "2019-02-23T17:42:59.429Z"}
```

Im Anfragetext muss der Ereignistyp immer EXTERN sein und der Objekttyp muss einen der folgenden Werte haben: *OPERATION*, *PRODUCTION_ORDER*, *WORKPLACE*, *STAFF_MEMBER*, *TOOL* oder kann leer sein.

Im Datenfeld können beliebig viele Informationen mit dem Ereignis übertragen werden. Der Ereignisname und die Objekt-ID sind optional.

Beispiel Java:

```
1.  BridgeAPI bridgeAPI = new BridgeAPI(url, user,secret);
2.  IEventClient eventClient = bridgeAPI.getEventClient();
3.
4.  final Map<String, String> data = new HashMap<>();
5.  data.put("myData1", "123");
6.  data.put("myData2", "456");
7.
8.  final EventWSModel eventData = new EventWSModel();
9.  eventData.setData(data);
10. eventData.setEventName("MY_EVENT");
11. eventData.setEventType(EventType.EXTERNAL);
12. eventData.setObjectId("4DDE054C036C4F8CAB424C7FB70E86B9");
13. eventData.setObjectType(EventObjectType.TOOL);
14.
15. final PostEventRequest request = new PostEventRequest(eventData);
16. eventClient.postEvent(request);
```

Beispiel CURL:

```
1.  curl -X POST https://$HOST:$PORT/ffwebservice/api/v2/events" -H "accept: */*" -H
    "authorization: Bearer 0ffd19e6-15a5-42fc-b080-86ccadacc168" -H "Content-Type:
    application/json" -d "{ \"data\": {\"myData1\" : \"123\", \"myData2\" : \"456\"},
    \"eventName\": \"MY_EVENT\", \"eventType\": \"INTERNAL\", \"objectId\":
    \"4DDE054C036C4F8CAB424C7FB70E86B9\", \"objectType\": \"TOOL\", \"timestamp\": \"2019-
    03-01T07:24:37.756Z\"}events
2.
3.      -H "accept: */*"
4.      -H "authorization: Bearer $TOKEN
5.      -H "Content-Type: application/json"
6.      -d "{
7.                \"data\": {\"myData1\" : \"123\", \"myData2\" : \"456\"},
8.                \"eventName\": \"MY_EVENT\",
9.                \"eventType\": \"EXTERNAL\",
10.               \"objectId\": \"4DDE054C036C4F8CAB424C7FB70E86B9\",
11.               \"objectType\": \"TOOL\",
12.               \"timestamp\": \"2019-03-01T07:24:37.756Z\"
13.           }
```

3.3 Dynamische Planung

Das folgende Video gibt einen Überblick darüber, wie FORCE Bridge API den Planungsalgorithmus, den Prognosealgorithmus und die Auswahl eines Planungsszenarios durch den Fertigungsplaner koordiniert. Details zu den einzelnen Applikationen finden Sie in den folgenden Abschnitten.

https://youtu.be/ugawxwQcgnA

Das Planungsergebnis - und mehr noch - das Prognoseergebnis stellt die organisatorische Interoperabilität aller Maschinen, IT-Systeme, Applikationen und Mitarbeiter in einer Fertigungsanlage sicher. Die gesamte Fertigungslogistik kann auf das Prognoseergebnis ausgerichtet werden, um die benötigten Materialien, Werkzeuge und sonstigen Ressourcen am richtigen Arbeitsplatz in der richtigen Qualität – Just-In-Time – bereitzustellen.

Die FORCE Bridge API selbst stellt die Interoperabilität der einzelnen Applikationen und IT-Systeme sicher, die mit der IoT-Plattform verbunden sind. Planungsszenarien können von Planungsalgorithmen bereitgestellt werden, ohne dass der Entwickler der Applikation den betriebswirtschaftlichen Hintergrund kennen muss, warum ein Vorgang einen oder mehrere Vorgänger innerhalb einer Vorgangsfolge oder eines Auftragsnetzes hat. Der Entwickler muss auch nicht wissen, warum bestimmte alternative Arbeitsplätze als Planungsalternativen mit unterschiedlichen Fertigungsvarianten definiert sind, wie die Rüstübergänge definiert sind oder wie der Personalbedarf an einem Arbeitsplatz ermittelt wird. Insbesondere kann jede Applikation unabhängig von anderen Applikationen entwickelt werden. Eine Applikation wird nur darüber informiert, dass die von ihr bereitgestellten Daten nicht mehr aktuell sind. Für den Planungsalgorithmus, der in regelmäßigen Zeitabständen neue Planungsszenarien ermittelt, zeigt das Ereignis *OBSOLETE_OPERATION_PLANNING_SCENARIOS* an, dass ein neues Planungsszenario bereitgestellt werden sollte. Für den Prognosealgorithmus, der in regelmäßigen Zeitabständen das aktuelle Prognoseergebnis ermittelt, zeigt das Ereignis *OBSOLETE_OPERATION_FORECAST_RESULTS* an, dass ein neues Prognoseergebnis bereitgestellt werden soll. Abgesehen davon ist der Entwickler nicht verpflichtet, die Existenz einer anderen Applikation zu kennen oder zu verstehen, wie sie funktioniert.

4 Anhang

4.1 Formeln

4.1.1 Laufzeiten

4.1.1.1 Arbeitsplatzbezogene Laufzeiten

4.1.1.1.1 Rüstzeit

Die Rüstzeit eines Arbeitsplatzes ist die Summe der Laufzeiten aller Betriebszustände, die zur Zeitbasis *SETUP* gehören. Die Summierung erfolgt für einen bestimmten Zeitraum auf einem Arbeitsplatz:

$$Rüstzeit = \frac{\sum}{Betriebszustand \in SETUP} Laufzeit$$

4.1.1.1.2 Produktionszeit

Die Produktionszeit eines Arbeitsplatzes ist die Summe der Laufzeiten aller Betriebszustände, die zur Zeitbasis *PRODUCTION* gehören. Die Summierung erfolgt für einen bestimmten Zeitraum auf einem Arbeitsplatz:

$$Produktionszeit = \frac{\sum}{Betriebszustand \in PRODUCTION} Laufzeit$$

4.1.1.1.3 Geplante Betriebszeit

Die geplante Betriebszeit ist die Summe der Laufzeiten aller Betriebszustände, die zur Zeitbasis *SCHEDULED_OPERATING_TIME* gehören. Die Summierung erfolgt für einen bestimmten Zeitraum über einen Arbeitsplatz:

$$Geplante\ Betriebszeit = \frac{\sum}{Betriebszustand \in SCHEDULED\ OPERATING\ TIME} Laufzeit$$

4.1.1.1.4 Belegungszeit

Die Belegungszeit eines Arbeitsplatzes ist die Gesamtdauer, die der Arbeitsplatz während seiner geplanten Betriebszeit für einen bestimmten Zeitraum von mindestens einem Vorgang belegt ist. Ein Vorgang belegt einen Arbeitsplatz während seiner Rüstphase und während seiner Bearbeitungsphase. Zeiträume außerhalb der planmäßigen Betriebszeit (z. B. Schichtpausen oder Freischichten) erhöhen die Belegungszeit des Arbeitsplatzes nicht, auch wenn in diesen Zeiträumen ein Vorgang am Arbeitsplatz eingeloggt ist. Die Belegungszeit eines Arbeitsplatzes ist die Summe der Ausführungszeiten aller Vorgänge, während sie im betreffenden Zeitraum an dem betreffenden Arbeitsplatz angemeldet waren.

$$Belegungszeit = \frac{\sum}{Operations} (Execution\ time)$$

ⓘ Bei der Kuppelproduktion werden nur die Ausführungszeiten der führenden Vorgänge berücksichtigt.

4.1.1.2 Vorgangsbezogene Laufzeiten

⚠ Die Laufzeiten müssen in der folgenden Reihenfolge ermittelt werden:

4.1.1.2.1 Rüstzeit

Die Rüstzeit eines Vorgangs ist die Rüstzeit des Arbeitsplatzes (wie oben definiert), während der Vorgang am Arbeitsplatz ausgeführt wurde. Ein Vorgang wird während seiner Rüst- und seiner Bearbeitungsphase ausgeführt. Wenn ein Vorgang an mehr als einem Arbeitsplatz ausgeführt wurde, ist die Rüstzeit des Vorgangs die Summe der Rüstzeiten all dieser Arbeitsplätze während der Ausführung des Vorgangs an jedem Arbeitsplatz.

ⓘ Auch während der Bearbeitungsphase eines Vorgangs kann sich der jeweilige Arbeitsplatz dennoch in einem Betriebszustand befinden, der der Zeitbasis SETUP zugeordnet ist, z. B. Zwischenrüstung während der Bearbeitungsphase.

4.1.1.2.2 Laufzeit der Rüstphase

Die Laufzeit eines einzelnen Rüstphasenzeitraums eines Vorgangs ist die Laufzeit aller planmäßigen Betriebszeiten des Arbeitsplatzes, an dem der Vorgang angemeldet wurde, vom Startdatum der Rüstphase bis zum Enddatum der Rüstphase. Zeiträume außerhalb der geplanten Betriebszeit (z. B. Schichtpausen) erhöhen nicht die Laufzeit der Rüstphase des Vorgangs. Befand sich ein Vorgang mehrfach in der Rüstphase (und das möglicherweise an mehreren Arbeitsplätzen), wird die Laufzeit der einzelnen Rüstphasen kumuliert, um die gesamte Laufzeit der Rüstphase zu berechnen.

ⓘ Da Rüstzeiten auch während der Bearbeitungsphase auftreten können, ist es möglich, dass die gesamte Laufzeit der Rüstphase kleiner ist als die Rüstzeit eines Vorgangs.

4.1.1.2.3 Laufzeit der Bearbeitungsphase

Die Laufzeit eines einzelnen Bearbeitungsphasenzeitraums eines Vorgangs ist die Laufzeit aller planmäßigen Betriebszeiten des Arbeitsplatzes, an dem der Vorgang angemeldet wurde, vom Starttermin der Bearbeitungsphase bis zum Endtermin der Bearbeitungsphase. Zeiträume außerhalb der terminierten Einsatzzeit erhöhen nicht die Laufzeit der Bearbeitungsphase des Vorgangs. Wenn sich ein Vorgang mehrmals in der Bearbeitungsphase befunden hat (und das kann an mehreren Arbeitsplätzen sein), wird die Laufzeit der einzelnen Bearbeitungsphasen kumuliert.

4.1.1.2.4 Ausführungszeit

Die Ausführungszeit eines Vorgangs ist die Gesamtzeit, die ein Vorgang an einem beliebigen Arbeitsplatz ausgeführt wurde. Da ein Vorgang während seiner Rüstphasen und seiner Bearbeitungsphasen ausgeführt wird, ist die Ausführungszeit eines Vorgangs die Summe der Laufzeit seiner Rüstphase und der Laufzeit seiner Bearbeitungsphase gemäß der obigen Definition, d. h.

$$Ausführungszeit = Laufzeit\ der\ Rüstphase + Laufzeit\ der\ Bearbeitungsphase$$

4.1.1.2.5 Bearbeitungszeit

Die Bearbeitungszeit eines Vorgangs ist seine Ausführungszeit abzüglich der Rüstzeit.

$$Ausführungszeit = Bearbeitungszeit - Rüstzeit$$

ⓘ Da Rüstzeiten auch während der Bearbeitungsphase anfallen können, ist es möglich, dass die Bearbeitungszeit eines Vorgangs kleiner ist als die Laufzeit der Bearbeitungsphase.

4.1.2 Leistungskennzahlen

4.1.2.1 Arbeitsplatzbezogene Leistungskennzahlen (KPIs)

Die Verfügbarkeit eines Arbeitsplatzes ist die Produktionszeit geteilt durch die planmäßige Betriebszeit des Arbeitsplatzes während des betreffenden Zeitraums. Die Verfügbarkeit eines Vorgangs ist die Produktionszeit geteilt durch die geplante Betriebszeit der Arbeitsplätze, an denen der Vorgang ausgeführt wurde, während der Ausführungszeit an diesen Arbeitsplätzen:

$$Verfügbarkeit = \frac{Produktionszeit}{geplante\ Betriebszeit}$$

Die Verfügbarkeit ist auch das Produkt der folgenden drei Leistungskennzahlen.

Der Belegungsgrad eines Arbeitsplatzes ist die Belegungszeit geteilt durch die geplante Betriebszeit des Arbeitsplatzes im betreffenden Zeitraum. Außerdem ist die Belegungszeit die Summe der Ausführungszeiten aller Vorgänge an dem jeweiligen Arbeitsplatz.

ⓘ Diese Summe kann von der Summe der Ausführungszeiten dieser Vorgänge abweichen, wenn einzelne Vorgänge teilweise an dem betreffenden Arbeitsplatz und teilweise an einem anderen Arbeitsplatz ausgeführt wurden.

$$Belegungsgrad = \frac{Belegungszeit}{geplante\ Betriebszeit} = \frac{\sum Vorgänge\ (Ausführungszeit)}{geplante\ Betriebszeit}$$

Die Rüstzeitreduktion eines Arbeitsplatzes ist die Summe der Bearbeitungszeiten aller Vorgänge an dem jeweiligen Arbeitsplatz geteilt durch die Belegungszeit des Arbeitsplatzes im jeweiligen Zeitraum.

ⓘ Die Bearbeitungszeiten am jeweiligen Arbeitsplatz können von den Gesamtbearbeitungszeiten dieser Vorgänge abweichen, wenn einzelne Vorgänge teilweise am jeweiligen Arbeitsplatz und teilweise an einem anderen Arbeitsplatz bearbeitet wurden.

$$Rüstreduktion = \frac{\sum Vorgänge\ (Bearbeitungszeit)}{Belegungszeit} = \frac{\sum Vorgänge\ (Bearbeitungszeit)}{\sum Vorgänge\ (Ausführungszeit)}$$

Die Prozessverfügbarkeit eines Arbeitsplatzes ist die Produktionszeit geteilt durch die Summe der Bearbeitungszeiten aller Vorgänge an dem jeweiligen Arbeitsplatz im betreffenden Zeitraum.

ⓘ Die Bearbeitungszeiten am jeweiligen Arbeitsplatz können von den Gesamtbearbeitungszeiten dieser Vorgänge abweichen, wenn einzelne Vorgänge teilweise am jeweiligen Arbeitsplatz und teilweise an einem anderen Arbeitsplatz bearbeitet wurden.

$$Prozessverfügbarkeit = \frac{Produktionszeit}{\sum_{Vorgänge}(Bearbeitungszeit)}$$

Die Leistung eines Arbeitsplatzes ist der folgende Quotient aus der Ausführungszeit der Vorgänge, die während des betreffenden Zeitraums an diesem Arbeitsplatz durchgeführt wurden:

$$Leistung = \frac{\sum_{Vorgänge} Sollzeit\ pro\ Einheit\ \times Outputmenge}{\sum_{Vorgänge}(Produktionszeit)}$$

Die Leistung eines einzelnen Vorgangs ist:

$$Leistung = \frac{Sollzeit\ pro\ Einheit\ \times\ Gesamt - Outputmenge}{Produktionszeit}$$

Der zeitliche Qualitätsgrad eines Arbeitsplatzes ist der folgende Quotient aus der Ausführungszeit der Vorgänge, die während des betreffenden Zeitraums an diesem Arbeitsplatz ausgeführt wurden:

$$Qualitätsgrad_{(zeitbasiert)} = \frac{\sum_{Vorgänge} Sollzeit\ pro\ Einheit\ \times Gutmenge}{\sum_{Vorgänge} Sollzeit\ pro\ Einheit\ \times Gesamt - Outputmenge}$$

Der mengenbezogene Qualitätsgrad eines Arbeitsplatzes ist der folgende Quotient:

$$Qualitätsgrad_{(mengenbasiert)} = \frac{Gutmenge}{Gesamt - Outputmenge}$$

Dies ist auch der Qualitätsgrad eines einzelnen Vorgangs. Für einen einzelnen Vorgang gibt es keinen Unterschied zwischen der zeitbasierten und der mengenbasierten Berechnung.

Die folgenden beiden Kennzahlen können sowohl auf die vergangene Erfassung als auch auf die Planung von Zukunftsszenarien angewendet werden:

Der Belegungsgrad einer Gruppe von Arbeitsplätzen während eines bestimmten Zeitraums:

$$Belegungsgrad = \frac{\sum_{Arbeitsplätze} Belegungszeit}{\sum_{Arbeitsplätze} geplante\ Betriebszeit}$$

Die Rüstreduktion einer Gruppe von Arbeitsplätzen während eines bestimmten Zeitraums:

$$Rüstreduktion = \frac{\sum_{Arbeitsplätze}(\sum_{Vorgänge}(Bearbeitungszeit))}{\sum_{Arbeitsplätze} Belegungszeit}$$

4.1.2.2 Auftragsbezogene Leistungskennzahlen

Die Planungseffizienz eines einzelnen Produktionsauftrages:

$$Planungseffizienz = \frac{\sum_{Vorgänge}(Ausführungszeit)}{Durchlaufzeit} \times \frac{1}{Parallelisierungsgrad}$$

Wobei die $\sum_{\text{Vorgänge}}(\text{Ausführungszeit})$ die Ausführungszeit der Produktion selbst ist.

Die Planungseffizienz einer Reihe von Fertigungsaufträgen, die innerhalb eines bestimmten Zeitraums fertiggestellt wurden oder deren Fertigstellung geplant ist:

$$Planungseffizienz = \frac{\sum_{Fertigungsaufträge}\left(Ausführungszeit \times \frac{1}{Parallelisierungsgrad}\right)}{\sum_{Fertigungsaufträge}(Durchlaufzeit)}$$

Die Einhaltung von Lieferterminen einer Reihe von Fertigungsaufträgen, die innerhalb eines bestimmten Zeitraums fertiggestellt wurden oder deren Fertigstellung geplant ist:

$$Einhaltung\ von\ Lieferterminen = \frac{Anzahl\ der\ rechtzeitlich\ abgeschlossenen\ Fertigungsaufträge}{Gesamtanzahl\ der\ abgeschlossenen\ Fertigungsaufträge}$$

4.2 Glossar

REST
Steht für Representational State Transfer und ist ein Programmierparadigma für verteilte Systeme. Die Idee dahinter ist, serverseitige Datenbankeinträge als Ressourcen anzubieten, die über einheitliche Befehlssätze erstellt, abgerufen, geändert oder gelöscht werden können.

Open API Spezifikation
Eine Spezifikation für maschinenlesbare Schnittstellen für die Beschreibung, Erzeugung, Nutzung und Visualisierung von RESTful Webservices.

Swagger UI
Ein Werkzeug zur Visualisierung und Interaktion mit Open-API-Ressourcen.

URI
Es steht für Uniform-Resource-Identifier und ist ein Bezeichner bestehend aus einer Zeichenkette, die zur Identifizierung einer abstrakten oder physischen Ressource verwendet wird. **RFC 3986**

UUID
Steht für Universally-Unique-Identifier und ist ein Standard für Bezeichner, die in der Softwareentwicklung verwendet werden.

ERP
Bedeutet Enterprise-Resource-Planning (Warenwirtschaftssystem) und beschreibt eine Softwarelösung zur Ressourcenplanung in einem Unternehmen. Ein ERP-System kann eine Datenquelle sein.

OEE
Bedeutet *Overall Equipment Effectiveness*.

OPE
Bedeutet *Overall Process Efficiency*.

Lazy loading
In der Softwareentwicklung bezeichnet *Lazy Loading* ein Entwicklungsmuster, bei dem Datenobjekte immer Werte oder andere abhängige Objekte bereitstellen, diese aber nur bei einer bestimmten Abfrage aus der Datenquelle abrufen.

MES
Steht für Manufacturing-Execution-System (Fertigungsleitsystem) oder monolithisch gekapseltes System, das Datenerfassungssysteme und die darauf arbeitenden Applikationen in einem monolithischen System vereint.