EasternGraphics
visualize your business

Die OFML–Schnittstellen *Article* und *CompositeArticle*

Dokumentversion 1.3

Thomas Gerth, EasternGraphics GmbH (Editor)

21. Oktober 2024

Rechtliche Hinweise

Inhaltsverzeichnis

Literatur

[oam] OAM – OFML Artikel-Mappings (OFML Part VI). Spezifikation Version 1.0.
Industrieverband Büro und Arbeitswelt e. V. (IBA)

[ocd] OCD – OFML Commercial Data (OFML Part IV). Spezifikation Version 4.3.
Industrieverband Büro und Arbeitswelt e. V. (IBA)

[oex] OEX – OFML Business Data Exchange (OFML Part VII). Spezifikation Version 3.1.
Industrieverband Büro und Arbeitswelt e. V. (IBA)

[ofml] OFML – Standardisiertes Datenbeschreibungsformat der Büromöbelindustrie.
Version 2.0, 3. überarbeitete Auflage.
Industrieverband Büro und Arbeitswelt e. V. (IBA)

[property] Die OFML–Schnittstelle *Property* (Spezifikation).
EasternGraphics GmbH

[rec20] Code List Recommendation 20 – Codes for Units of Measure Used in International Trade.
Revision 17 (Annexes I to III), 2021.
The United Nations Economic Commission for Europe (UNECE)
(https://unece.org/trade/uncefact/cl-recommendations)

Die Dokumente (außer [rec20]) sind über das Download Center von EasternGraphics

https://download-center.pcon-solutions.com

in der Kategorie *OFML Specifications* verfügbar.

1 Einleitung

Die Schnittstelle *Article* definiert alle Funktionen, die ein Typ implementieren muß, dessen Instanzen einen kaufmännischen Artikel repräsentieren.
(Instanzen, die einen Artikel repräsentieren, werden im Folgenden auch als *Artikelinstanz* bezeichnet.)

Die Spezifikationen in diesem Dokument ersetzen bzw. aktualisieren die Schnittstellen–Spezifikation in [ofml]!

Ein *kompositer Artikel* besteht aus einer festen oder variablen Anzahl von Unterartikeln. Ein Unterartikel eines kompositen Artikels kann selber auch ein kompositer Artikel sein[1].
Die Schnittstelle *CompositeArticle* definiert die Funktionen, die ein Typ implementieren muß, dessen Instanzen einen kompositen Artikel repräsentieren. Sie erweitert die Schnittstelle *Article*, d.h., der entsprechende Typ muss auch alle Funktionen der Schnittstelle *Article* implementieren!

Inhaltlich zusammengehörende Funktionen werden zu einer Methodengruppe zusammengefaßt und in entsprechenden Unterabschnitten der Abschnitte 2 (Schnittstelle *Article*) und 3 (Schnittstelle *CompositeArticle*) spezifiziert.

Anhang A enthält einen alphabetischen Index der Funktionen.

Der Basistyp *OiPlElement* hat für alle Funktionen der Schnittstelle *Article* eine Default-Implementierung. (Wo erforderlich/hilfreich, wird das Verhalten der Default-Implementierung explizit beschrieben.)
Der Basistyp *OiPlElement* kann jedoch nicht direkt für eine Artikelinstanz verwendet werden, da er keine generische Geometrie-Erzeugung bietet. Siehe dafür z.B. den von *OiPlElement* unmittelbar abgeleiteten Basistyp *OiOdbPlElement*.

Der Basistyp *OiPart* ist konzeptionell nicht für Artikelinstanzen vorgesehen. Da in der Vergangenheit dennoch davon abgeleitete Typen für diesen Zweck verwendet wurden, hat auch *OiPart* für alle Funktionen eine Default-Implementierung.

Die Implementierungen der Funktion *isCat()* (Schnittstelle *MObject*) in den Basistypen *OiPlElement* und *OiPart* liefert für die Schnittstellen-Kategorie `@IF_Article` den Wert 1 (wahr).
In abgeleiteten Klassen kann *isCat()* jedoch überschrieben sein und dann für die Kategorie `@IF_Article` den Wert 0 liefern, d.h., diese Instanz ist dann nicht als Artikelinstanz zu betrachten. Für Clienten bedeutet dies, dass auf dieser Instanz dann auch keine Methoden dieser Schnittstelle aufgerufen werden dürfen[2].

Der Anhang B enthält Spezifikationen von vordefinierten Standard–Kategorien für Artikelinstanzen.

[1] Das zugrundeliegende Software-Entwurfsmuster ist *Composite*.
[2] Falls doch Methoden gerufen werden, führt das im einfacheren Fall nur zu einer schlechteren Performanz, im schwereren Fall kann das jedoch auch zu einem Fehlverhalten führen.

2 Die Schnittstelle *Article*

Funktionen, die in der Schnittstellen–Spezifikation in [ofml] (noch) nicht beschrieben wurden, sind mit **new** gekennzeichnet.

Funktionen, deren Spezifikation im Vergleich zu [ofml] modifiziert wurde, sind mit **mod** gekennzeichnet.

Einige Funktionen haben einen Parameter *pLanguage* vom Typ *String*, mit dem die gewünschte Sprache für textuelle Bestandteile des Rückgabewertes spezifiziert werden kann. Die Sprache ist als zweistelliges Kürzel gemäß ISO 639-1 anzugeben, z.B.: `de` (Deutsch) und `en` (Englisch).

2.1 Programm–Zugehörigkeit

- *getProgram()* → *Symbol*

  Die Funktion liefert die ID des OFML–Programms, dem die implizite Instanz angehört.

  Das Programm definiert, u.a., welche in der globalen Planungsinstanz (Basistyp *OiPlanning*) registrierte Instanz von *OiProgInfo* zur Erledigung allgemeiner programm-bezogener Aufgaben herangezogen wird.
  Des Weiteren definiert das Programm, welche im globalen Produktdatenmanager (Basistyp *OiPDManager*) registrierte Produktdatenbank (Instanz von *OiProductDB*) die kaufmännischen Daten für den durch die implizite Instanz repräsentierten Artikel enthält.

  Konzeptionell gehört diese Funktion zur Schnittstelle *Base*[3], aufgrund der Verwendung bei der Ermittlung der relevanten Produktdatenbank (s.o.) hat die Programm-Zugehörigkeit für Artikelinstanzen jedoch eine besondere Bedeutung.

  Die Standardimplementierung der Schnittstelle *Base* delegiert die Anfrage an das nächste in der Objekthierachie weiter oben angesiedelte Planungselement (Instanz von *OiPlElement*).

  Die Standardimplementierung in *OiPlElement* basiert auf einer Membervariablen, welche während der Erzeugung und Initialisierung der Artikelinstanz initialsiert wird (s.a. Anh. E).

2.2 Initialisierung und Artikelcodes

- *setArticleSpec(pSpec(String))* → *Void* **mod**

  Die Funktion weist der impliziten Instanz eine Artikelnummer zu.

  Die Artikelnummer ist ein alphanumerischer Code, der ein Produkt (Artikel) des Herstellers innerhalb des führenden Produktions- und Planungssystems (PPS) eindeutig identifiziert. Handelt es sich um einen konfigurierbaren Artikel, wird die Artikelnummer auch als *Grundartikelnummer* bezeichnet, im Gegensatz zu einer evtl. zusätzlich spezifizierten erweiterten Endartikelnummer, die das auskonfigurierte Produkt identifiziert (s. *setXArticleSpec()* unten).

  Im Ergebnis der Funktion soll die Instanz die Merkmale (Properties) besitzen, die in der zuständigen Produktdatenbank[4] für die initiale (Grund-)Konfiguration des Artikels definiert sind.

[3] und wird bei einer Überarbeitung der OFML-Spezifikation in [ofml] auch dort beschrieben werden
[4] Die zuständige Produktdatenbank ergibt sich aus der Programm-Zugehörigkeit der impliziten Instanz, s. 2.1.

Ob im Ergebnis der Funktion auch schon die entsprechende geometrische Repräsentation vorliegt, hängt von der Implementierung im konkret verwendeten Typ ab. Die Erzeugung der geometrischen Repräsentation kann aus Performzgründen auch erst im nachfolgenden Aufruf der Funktion *setXArticleSpec()* erfolgen (s.u.), mit dem der impliziten Instanz ein (möglicherweise leerer) Variantencode zugewiesen wird.

> Die Implementierung im Basistyp *OiOdbPlElement* z.B. generiert nur die initialen Merkmale unter Verwendung der Methode *setupProps()* des globalen Produktdatenmanagers (Basistyp *OiPDManager*).

Die Funktion hat keinen Effekt, wenn sie zwischen den Persistezregeln `START_EVAL` und `FINISH_EVAL` für die globale Planungsinstanz gerufen wird.

- *getArticleSpec()* → *String* | *Void* **mod**

  Die Funktion liefert die (Grund-)Artikelnummer des Artikels, der durch die implizite Instanz repräsentiert wird bzw. einen Wert vom Typ *Void*, falls keine Artikelspezifikation für die implizite Instanz vorliegt.

  Ist das Ergebnis der Funktion ein Wert vom Typ *Void*, wird für die Instanz kein Eintrag in der Warenkorb-Struktur der Applikation angelegt.

  Die Standardimplementierung delegiert die Anfrage an die Methode *object2Article()* des globalen Produktdatenmanagers (Basistyp *OiPDManager*), wobei die implizite Instanz als Argument übergeben wird.

  Aus Gründen der Performanz sollten konkrete Typen, deren Instanzen einen Artikel repräsentieren, diese Implementierung überschreiben und den Wert einer entsprechenden Membervariablen liefern, an welche die Artikelnummer zugewiesen ist, die an die Funktion *setArticleSpec()* übergeben wurde (s.o.). Dieser Grundsatz wird z.B. in der Implementierung im Basistyp *OiOdbPlElement* angewendet.

- *getArticleParams()* → *Any* **mod**

  Die Funktion liefert die Parameter der impliziten Instanz, die neben dem Typ der Instanz zur Ermittlung der Artikelnummer verwendet werden sollen.

  Die Funktion ist nur für Typen relevant, die keine eigene Implementierung der Funktion *getArticleSpec()* unter Verwendung einer entsprechenden Membervariablen besitzen (s.o.), und deren Instanzen unterschiedliche Artikel repräsentieren können. Die Funktion *getArticleParams()* wird dann im Rahmen der Methode *object2Article()* des globalen Produktdatenmanagers (Basistyp *OiPDManager*) gerufen.

  Rückgabewert ist ein Vektor mit den Parameterwerten oder ein String, der bereits die in das jeweilige Persistenzformat konvertierten Parameterwerte enthält. Werden für die Ermittlung der Artikelnummer keine Parameter benötigt, liefert die Funktion einen Wert vom Typ *Void*.

  Die Standardimplementierung liefert einen Wert vom Typ *Void*.

  Die Implementierung im Basistyp *OiPlElement* delegiert an die gleichnamige Methode des globalen Produktdatenmanagers (Basistyp *OiPDManager*), wobei die implizite Instanz als Argument übergeben wird. Der globale Produktdatenmanager verwendet dann externe Mappingtabellen, um die Parameter zu ermitteln. Die Mappingtabellen sind im OFML Part VI (OAM) standardisiert [oam].

- *setXArticleSpec(pType(Symbol), pSpec(String)) → Void* **mod**

  Die Funktion weist der impliziten Instanz eine Artikelspezifikation des angegebenen Typs zu.

  Folgende Spezifikationstypen sind definiert:

  `@Base`
  : Grundartikelnummer: eindeutige Kennung des Herstellers für den Artikel ohne Bezug auf eine konkrete Ausführung/Konfiguration.

  `@VarCode`
  : Hersteller-spezifischer Variantencode: beschreibt die konkrete Ausführung/Konfiguration des Artikels.

  `@OFMLVarCode`
  : Hersteller-unabhängiger Variantencode, auch bezeichnet als *OFML–Variantencode*: beschreibt die konkrete Ausführung/Konfiguration des Artikels.

    Der OFML–Variantencode sollte von den OFML–Applikationen zur Wiederherstellung von gespeicherten Artikelkonfigurationen verwendet werden (s. Abschn. 2.6). Sein Aufbau ist im Anhang C beschrieben.

  `@Final`
  : Hersteller-spezifische Endartikelnummer: kennzeichnet den Artikel und beschreibt seine konkrete Ausführung/Konfiguration.

    > Normalerweise setzt sich die Endartikelnummer aus der Grundartikelnummer und dem Variantencode zusammen. Dies ist jedoch abhängig von dem zugrundeliegenden Produktdatensystem.

  Bei Übergabe einer Artikelspezifikation des Typs `@Base` verhält sich die Funktion wie die Funktion *setArticleSpec()*, siehe oben.

  Die kaufmännische Initialisierung einer Artikelinstanz erfolgt durch unmittelbar aufeinanderfolgende Aufrufe von *setArticleSpec()* (bzw. *setXArticleSpec()* mit Spezifikationstyp `@Base`) und *setXArticleSpec()* mit Spezifikationstyp `@VarCode`.

  Der übergebene Variantencode kann ein leerer String sein. Die Artikelinstanz behält dann die während *setArticleSpec()* erzeugte initiale (Grund-)Konfiguration.

  Der übergebene Variantencode kann teilbestimmt sein, also nur die Merkmale codieren, die abweichend von der Grundkonfiguration bewertet werden sollen.

  Wird ein Variantencode bzw. eine Endartikelnummer übergeben, die nicht zu dem Artikel passen, der durch die implizite Instanz repräsentiert wird, behält die Instanz die bis dahin erzeugte Konfiguration bzw. es wird nur ein Teil der in der übergebenen Spezifikation codierten Merkmale neu bewertet.

  Bei Zuweisung eines Variantencodes oder einer Endartikelnummer besitzt die implizite Instanz im Ergebnis der Funktion auch die geometrische Repräsentation entsprechend der erzeugten Artikelkonfiguration.

- *getXArticleSpec(pType(Symbol)) → String | Void* **mod**

  Die Funktion liefert die Spezifikation des geforderten Typs für den Artikel, der durch die implizite Instanz repräsentiert wird.

  Falls keine Artikelspezifikation des geforderten Typs für die implizite Instanz vorliegt, liefert die Funktion einen Wert vom Typ *Void*.

  Die möglichen Spezifikationstypen sind bei der Funktion *setXArticleSpec()* beschrieben (s.o.).

  Bei Übergabe einer Artikelspezifikation des Typs `@Base` entspricht der Rückgabewert dem der Funktion *getArticleSpec()* (s.o.).

  Die Standardimplementierung delegiert die Anfrage beim Spezifikationstyp `@Base` an die Methode *getArticleSpec()* und bei allen anderen Spezifikationstypen an die gleichnamige Methode des globalen Produktdatenmanagers (Basistyp *OiPDManager*), wobei die implizite Instanz und der Spezifikationstyp als Argumente übergeben werden.

2.3 Sonstiger Zustand

- *getObjState(pStateType(Symbol)) → Any* **new**

  Die Funktion liefert den aktuellen Wert der impliziten Instanz bezüglich des Zustands des angegebenen Typs.

  Diese Funktion ist bereits für die Basisschnittstelle *Base* definiert. Die Standardimplementierung der Schnittstelle *Article* überschreibt die von der Schnittstelle *Base* geerbte Implementierung wie folgt:
  Sie behandelt die Zustandstypen `@OI_UpdateState` und `@OI_ConsistencyState` (siehe unten) und ruft für alle anderen Zustandstypen die geerbte Implementierung[5].

  Der Zustandstyp `@OI_UpdateState` bezieht sich auf den Aktualisierungsstatus, der die Nutzbarkeit der aktuell installierten Produktdatenbank definiert.
  Die möglichen Werte (vom Typ *Symbol*) für den Aktualisierungsstatus und die während der Aktualisierung von Artikelinstanzen verwendeten Funktionen sind im Abschn. 2.6 beschrieben.

  Der Zustandstyp `@OI_ConsistencyState` bezieht sich auf den Konsistenzstatus, der das Ergebnis des letzten Aufrufs der Methode *checkConsistency()* enthält, siehe Abschn. 2.7.
  Direkt nach der Erzeugung der Artikelinstanz ist der Konsistenzstatus undefiniert, was mit einem Wert vom Typ *Void* ausgedrückt wird.
  Der Konsistenzstatus wird in einer Membervariablen der Artikelinstanz gespeichert und wird als Ergebnis von *checkConsistency()* verwendet (anstatt die Prüfung tatsächlich durchzuführen), wenn die Artikelinstanz nach dem Laden aus einer Dump-Repräsentation (s. Abschn. 2.6) noch nicht aktualisiert wurde bzw. nicht aktualisiert werden konnte (siehe Zustandstyp `@OI_UpdateState` oben).

- *isUp2Date() → Int* **new**

  Die Funktion gibt den Wert 1 (wahr) zurück, wenn die aktuell installierte Produktdatenbank verwendet werden darf, um die Konfiguration der impliziten Artikelinstanz zu ändern.

  Das Ergebnis ergibt sich aus dem aktuellen Aktualisierungsstatus gemäß Funktion *getObjState()* (s.o.) für den Zustandstyp `@OI_UpdateState`: Der Rückgabewert ist 1, wenn der aktuelle Aktualisierungsstatus `@Up2Date` ist, anderenfalls 0.

- *setObjState(pStateType(Symbol), pValue(Any)) → Void* **new**

  Die Funktion weist der impliziten Instanz einen neuen Wert bezüglich des Zustands des angegebenen Typs zu.

  Diese Funktion ist bereits für die Basisschnittstelle *Base* definiert. Die Standardimplementierung der Schnittstelle *Article* überschreibt die von der Schnittstelle *Base* geerbte Implementierung wie folgt:
  Sie behandelt die Zustandstypen `@OI_UpdateState` und `@OI_ConsistencyState` (s.o. bei der Funktion *getObjState()*) und ruft für alle anderen Zustandstypen die geerbte Implementierung.

  Beim Zustandstyp `@OI_UpdateState` führt die Standardimplementierung bei einem geänderten Zustandswert neben der Zuweisung des Wertes an eine Membervariable noch folgende Aktionen durch:

  - Ist der neue Zustand `@Migratable` oder `@Invalid` (s. Anhang D), werden alle aktuell editierbaren Properties deaktiviert (siehe Methode *invalidateProperties()* in der Schnittstelle *Property*). Die Artikelinstanz kann damit in diesen Zuständen nicht (mehr) konfiguriert werden.
  - Ist der neue Zustand `@Up2Date` (s. Anhang D), wird an den globalen ChangeManager (Basistyp *OiChangeManager*) ein Event vom Typ `@ArticleUpdated` ausgegeben, wobei die implizite Instanz als Publisher und der alte Zustand als Event-Argument übermittelt werden.

[5] Diese behandelt Zustandstypen, die sich auf Flags beziehen, die in der OFML–Laufzeitumgebung Objekt-bezogen verwaltet werden.

- *getAddStateCode(pDomain(Void | Symbol)) → String | Vector* **new**

  Die Funktion liefert einen Code, der den Zustand der internen Variablen der impliziten Instanz beschreibt, die sich vom Zustand direkt nach Erzeugung und kaufmännischen Initialisierung der Artikelinstanz unterscheiden[6].

  Dieser Code wird primär von OFML–Applikationen benötigt und abgerufen, die den Zustand von Artikelinstanzen in einer persistenten Warenkorbstruktur speichern, siehe Abschn. 2.6.

  Der von dieser Funktion gelieferte Code sollte keine Repräsentation der aktuellen Artikelkonfiguration enthalten, da diese bereits von der Funktion *getXArticleSpec()* geliefert wird (s. Abschn. 2.2).

  Wenn der Parameter *pDomain* ein Wert vom Typ *Void* ist, muss der komplette zusätzliche Zustandscode geliefert werden. Der Code wird dann in separate Abschnitte für verschiedene Domänen aufgeteilt und der Rückgabewert ist ein Vektor von Vektorpaaren, die jeweils den Zustand einer spezifischen Domäne beschreiben:

  1. Domäne (Symbol)
  2. Code (String)

     Bsp.: `[[@Domain1, "Code1"],[@Domain2, "Code2"], ...]`

  OFML–Applikationen verwenden diese Variante des Funktionsaufrufs, wenn eine persistente Warenkorb–Repräsentation der Artikelinstanz erstellt wird.

  Wenn ein Client Kenntnis über spezifische Domänen hat und den zusätzlichen Zustandscode nur für eine bestimmte Domäne abrufen möchte, kann diese im Parameter *pDomain* angegeben werden. In diesem Fall ist der Rückgabewert eine Zeichenkette, die nur den Code für die angeforderte Domäne enthält[7], oder eine leere Zeichenkette, wenn die Domäne nicht vom Typ der impliziten Instanz unterstützt wird.

  Die Standardimplementierung liefert einen leeren Vektor, wenn der Parameter *pDomain* ein Wert vom Typ *Void* ist, andernfalls eine leere Zeichenkette.

  Wenn diese Funktion in einer abgeleiteten Klasse überschrieben wird (zusammen mit der zugehörigen Funktion *setAddStateCode()*, s.u.), sollten die folgenden Regeln beachtet werden:

  – Wenn der Parameter *pDomain* ein Wert vom Typ *Void* ist, muss zuerst die geerbte Implementierung aufgerufen werden. Dann muss der eigene Domänenabschnitt hinzugefügt werden (falls vorhanden).
  – Wenn der Parameter *pDomain kein* Wert vom Typ *Void* ist, wird der entsprechende Code zurückgegeben, wenn die Domäne von der Klasse unterstützt wird. Andernfalls muss das Ergebnis der geerbten Implementierung zurückgegeben werden.
  – Um Namenskonflikte bei Domänenbezeichnern zu vermeiden, wird empfohlen, den Namen der Klasse selbst für den Abschnitt zu verwenden, den die Klasse in den Code einfügt.
    Die Domäne `@ChildProps` ist für Metatypen reserviert.

  Bei Bedarf kann eine Klasse/Implementierung auch mehrere Domänen definieren.

- *setAddStateCode(pDomain(Void | Symbol), pCode(Vector | String)) → Void* **new**

  Die Funktion weist der impliziten Instanz den angegebenen zusätzlichen Zustandscode zu.
  Dieser Code beschreibt den Zustand der internen Variablen der impliziten Instanz, die sich vom Zustand direkt nach Erzeugung und kaufmännischen Initialisierung der Artikelinstanz unterscheiden.

  Wenn der Parameter *pDomain* ein Wert vom Typ *Void* ist, wird der komplette zusätzliche Zustandscode in Form eines Vectors übergeben, der Angaben zu allen relevanten Domänen enthält. Zum Aufbau dieses Vectors siehe Spezifikation der Funktion *getAddStateCode()* oben.

  Wenn der Parameter *pDomain kein* Wert vom Typ *Void* ist, wird nur der Code (String) übergeben, der für diese Domäne relevant ist.

[6] Daher wird dieser Code als zusätzlicher Zustandscode bezeichnet.
[7] z.B. `"Code1"` für die Domäne `@Domain1`

Es wird erwartet, dass der Client denselben Wert für den Parameter *pDomain* verwendet wie auch beim Aufruf von *getAddStateCode()*, mit dem der übergebene Code ermittelt wurde.

OFML–Applikationen verwenden die Funktion mit einem Wert vom Typ *Void* für den Parameter *pDomain*, wenn eine Artikelinstanz aus einer gespeicherten Warenkorb–Repräsentation wieder hergestellt wird. Der Aufruf erfolgt dabei, nachdem die gespeicherte kaufmännische Konfiguration mittels Aufruf von *setupConfiguration()* (s. Abschn. 2.6) wieder hergestellt wurde[8].

Die Standardimplementierung ist leer, d.h. es werden keine Aktionen ausgeführt.

Wenn diese Methode in einer abgeleiteten Klasse überschrieben wird (zusammen mit der zugehörigen Methode *getAddStateCode()*, s.o.), sollten die folgenden Regeln beachtet werden:

- Wenn der Parameter *pDomain* ein Wert vom Typ *Void* ist, muss zuerst der von der Klasse zu behandelnde Abschnitt aus dem übergebenen Code extrahiert werden. Dann muss die geerbte Implementierung unter Übergabe des (möglicherweise) gekürzten Vektors aufgerufen werden. Schließlich muss der Code für die eigene (extrahierte) Domäne verarbeitet werden (falls vorhanden).
- Wenn der Parameter *pDomain kein* Wert vom Typ *Void* ist, wird der übergebene Code verarbeitet, wenn die Domäne von der Klasse unterstützt wird. Andernfalls muss die geerbte Implementierung aufgerufen werden.

2.4 Produktdaten

Vorbemerkung:
Zu den Produktdaten gehören konzeptionell auch Eigenschaftsbeschreibungen und Variantentexte. Da zum Abrufen dieser Informationen diverse Funktionen zur Verfügung stehen, werden diese in einem eigenen Abschnitt (2.5) behandelt.

Verschiedene Artikelinstanzen können dieselbe Konfiguration eines Artikels repräsentieren. Konzeptionell sollen die in diesem und dem nachfolgenden Abschnitt beschriebenen Funktionen für dieselbe **Artikelkonfiguration** auch die gleichen Resultate liefern (unter Berücksichtigung eines ggf. vorhandenen Sprach–Parameters).

Zwei Artikelinstanzen repräsentieren dieselbe Konfiguration, wenn folgende Angaben übereinstimmen:

- die Programm-Zugehörigkeit, siehe Funktion *getProgram()* (Abschn. 2.1)
- die (Grund-)Artikelnummer, siehe Funktion *getArticleSpec()* (Abschn. 2.2)
- der OFML–Variantencode, siehe Funktion *getXArticleSpec()* (Abschn. 2.2)

Applikationen können diese Angaben zur Erzeugung eines Schlüssels für *Produktdaten–Caches* nutzen, um die Anzahl von Methodenaufrufen zu minimieren und somit die Performanz zu verbessern. Wird so ein Cache auch für die Preisinformationen benutzt, muss der Schlüssel zusätzlich auch das Preisdatum kodieren, siehe Funktion *getPriceDate()* (Abschn. 2.7).

- *getArticlePrice(pLanguage(String), ...)* → *Any[]* | *Void* **mod**

  Die Funktion liefert Preisinformationen für die implizite Artikelinstanz.

  Der Parameter *pLanguage* gibt die Sprache an, die für textuelle Bestandteile der Preisinformationen verwendet werden soll.

  Ist ein weiterer, optionaler Parameter vom Typ *String* angegeben, so spezifiziert dieser die gewünschte Währung. Die Preisinformationen müssen von der Funktion jedoch nicht in dieser Währung geliefert werden (wenn z.B. die zugrundeliegende Produktdatenbank keine Preise in dieser Währung bereitstellen kann). In diesem Fall muß der Client anhand von Umtauschsätzen gegebenenfalls selber eine Umrechnung in die gewünschte Währung vornehmen.

[8] Deswegen gilt, analog zur Festlegung bei der Funktion *getAddStateCode()*, dass der übergebene Code keine Repräsentation von konfigurierbaren Merkmalen des Artikels enthalten sollte.

Sind in den Produktdaten keine Preisinformationen für den Artikel hinterlegt, liefert die Funktion einen Wert vom Typ *Void*.
Anderenfalls ist der Rückgabewert eine Liste, die die einzelnen *Preiskomponenten* bzw. den Endpreis beinhaltet.

Jeder Eintrag in der Rückgabeliste ist ein Vektor aus 3, optional 5 Elementen:

1. eine Beschreibung (*String*), die die Art bzw. den Existenzgrund der Preiskomponente spezifiziert, z.B. den Grund eines Aufpreises
2. der Verkaufspreis der Preiskomponente (*Float*)
3. der Einkaufspreis der Preiskomponente (*Float*)
4. (optional) der Bezeichner der Variantenkondition der Preiskomponente (*Void* | *String*)
5. (optional) der Faktor, der bei der Berechnung des Betrages der Preiskomponente auf den Betrag aus der Preisdatenbank angewendet wurde (*Void* | *Float*)

Eine Ausnahme bildet der erste Eintrag, der anstelle der Preise die verwendete Währung (*String*) für den Verkaufspreis (Element 2) und den Einkaufspreis (Element 3) enthält. (Die anderen Elemente in diesem Eintrag haben keine Bedeutung.)

Der letzte Eintrag der Liste spezifiziert den (akkumulierten) Endpreis. Die optionalen Einträge dazwischen spezifizieren die einzelnen Preiskomponenten (Basispreis, Aufpreise, Rabatte usw.). Enthält eine solche Preiskomponente im ersten Element die Beschreibung `"@baseprice"`, so ist sie explizit als Basispreis gekennzeichnet.

Die Elemente 4 und 5 sind nur dann in der Rückgabestruktur enthalten, wenn diese Aspekte bei der Preisermittlung in der zuständigen Produktdatenbank[9] relevant sind.

Die Elemente 4 und 5 im Eintrag für den Endpreis sind nur dann relevant, wenn in der Rückgabeliste keine sonstigen Preiskomponenten enthalten sind und der Endpreis gleich dem dem Basispreis ist.

Ist der Endpreis-Betrag für eine Preisart (Verkaufspreis vs. Einkaufspreis) 0.0 und ist im ersten Listeneintrag für diese Preisart ein Leerstring für die entsprechende Währung angegeben, so bedeutet dies, dass für den Artikel in den Produktdaten keine Angaben zu dieser Preisart enthalten sind.

Unterstützt die zuständige Produktdatenbank bei der Preisermittlung Gültigkeitszeiträume für Preiskomponenten, wird das Datum verwendet, welches durch die Methode *getPriceDate()* für die implizite Instanz geliefert wird (s. Abschn. 2.7).

Sind in den Produktdaten zu diesem Datum keine Preisinformationen für den Artikel hinterlegt, liefert die Funktion folgenden Rückgabewert:

```
@(["@invalid_date", NULL, NULL], ["invalid price date", NULL, NULL])
```

Die Ressource `@invalid_date` im 1. Listeneintrag kann mit einem konkreten Paketbezeichner qualifiziert sein. Die Bezeichnung des Zustands im 2. Listeneintrag sollte in der im Parameter *pLanguage* angeforderten Sprache angegeben sein.

Die OFML–Laufzeitumgebung muss in diesem Fall (zusätzlich) dafür sorgen, dass die Funktion *checkConsistency()* (s. Abschn. 2.7) für die implizite Instanz eine entsprechende Inkonsistenz liefert.

Die Standardimplementierung delegiert an die gleichnamige Methode des globalen Produktdatenmanagers (Basistyp *OiPDManager*), wobei die implizite Instanz, die angeforderte Sprache und die ggf. gewünschte Währung als Argumente übergeben werden.

[9] Die zuständige Produktdatenbank ergibt sich aus der Programm-Zugehörigkeit der impliziten Instanz, s. 2.1.

- *getArticleText(pLanguage(String), pForm(Symbol)) → String[] | Void* **mod**

  Die Funktion liefert in der angegebenen Sprache für den Artikel, der durch die implizite Instanz repräsentiert wird, eine textuelle Beschreibung der gewünschten Form.

  Mögliche Werte für den Paramter *pForm* sind:

  `@l` Langbeschreibung (Artikellangtext)

  Der Artikellangtext sollte alle wesentlichen festen (nicht-konfigurierbaren) Eigenschaften des Artikels beschreiben[10].
  Der Artikellangtext ist in der Regel mehrzeilig.

  `@s` Kurzbeschreibung (Artikelkurztext)

  Der (eher technisch orientierte) Artikelkurztext ist oft eine abgekürzte Fassung des Langtextes.
  Der Artikelkurztext sollte einzeilig sein.

  `@pn` Produktname

  Der (eher Marketing-getriebene) Produktname kann in Anwender-orientierten Produktansichten (anstelle des Artikelkurztextes) verwendet werden.
  Der Produktname ist in der Regel einzeilig.

  Rückgabewert ist eine Liste von Zeichenketten, die die einzelnen Zeilen der Beschreibung enthalten bzw. ein Wert vom Typ *Void*, falls keine Beschreibung in der gewünschten Form und in der angegebenen Sprache vorliegt.

  Die Standardimplementierung delegiert an die gleichnamige Methode des globalen Produktdatenmanagers (Basistyp *OiPDManager*), wobei die implizite Instanz, die geforderte Sprache und die gewünschte Form als Argumente übergeben werden[11].

- *getArticleClassifications(pLanguages(String[] | Void)) → Any | Void* **new**

  Die Funktion liefert Informationen zur Klassifizierung des Artikels, der durch die implizite Instanz repräsentiert ist.

  Im Parameter *pLanguages* können die Sprachen (Codes gemäß ISO 639-1 Alpha 2) spezifiziert werden, für die sprach-spezifische Beschreibungen der Klassen geliefert werden sollen. Hat der Parameter einen Wert vom Typ *Void*, werden keine sprach-spezifischen Beschreibungen geliefert. Wenn der Parameter eine leere Sequenz (*List* oder *Vector*) enthält, werden alle in der Produktdatenbank hinterlegten sprach-spezifischen Beschreibungen geliefert.

  Wenn für den Artikel keine Informationen zur Klassifizierung vorliegen, wird ein Wert vom Typ *Void* geliefert, anderenfalls ein (nicht-lererer) *Vector* von Klassifikationsinformationen. Eine *Klassifikationsinformation* beschreibt je eine Klassifizierung nach einem konkreten Klassifikationssystem.

[10] Eine Beschreibung der aktuellen Werte der veränderbaren/konfigurierbaren Merkmale des Artikels wird durch die Funktion *getArticleFeatures()* geliefert (siehe Abschn. 2.5).

[11] Die Standardimplementierung in *OiPDManager* wiederum delegiert an die gleichnamige Methode der zuständigen Produktdatenbank (Basistyp *OiProductDB*).

Eine Klassifikationsinformation ist ein *Vector* aus folgenden Elementen:

1. Name/Bezeichner des Klassifikationssystems, ohne Versionsangabe o.ä. (*String*).

   Für branchen- und unternehmensübergreifende Standards sind aktuell folgende Bezeichner vordefiniert:

   `ECLASS` Klassifizierung gemäß Standard ECLASS[12]

   `UNSPSC` Klassifizierung gemäß Standard UNSPSC[13]

   Bezeichner für sonstige Klassifikationssysteme beginnen mit dem Zeichen `@`. Bezeichner für hersteller-spezifische Klassifikationssysteme bestehen aus dem Zeichen `@`, gefolgt vom kaufmännischen Herstellerkürzel.

2. Qualifier des Systems, z.B. die Version (*String*).

   Die Zeichenkette kann leer sein, wenn in den Produktdaten kein spezieller Qualifier hinterlegt ist.
   Semantik und Syntax des Qualifiers hängen vom zugrundeliegenden Produktdatenformat ab. Für die Formatversion 4.3 von OCD [ocd] z.B. gilt:

   - Beim Standard ECLASS ist der Qualifier die Versionsnummer im Format `x.y`.
   - Bei einem hersteller-spezifischen System ist der Qualifier der Teil aus dem OCD–Systembezeichner nach Herstellerkürzel und Unterstrich.

3. Identifier der Klasse (*String*).

4. Sprach-spezifische Beschreibung(en) der Klasse (*Any[]|Void*).

   Hat der Parameter *pLanguages* einen Wert vom Typ *Void*, so hat auch dieses Element einen Wert vom Typ *Void*. Anderenfalls ist das Element ein *Sprache–Text–Mapping*, bestehend aus einer Sequenz (*Vector* oder *List*) aus null oder mehr Elementen, wobei jedes Element wiederum ein *Vector* aus zwei Elementen ist:

   1. Sprachcode gemäß ISO 639-1 Alpha-2 (*String*)
   2. Text/Beschreibung in der entsprechenden Sprache (*String*)

   Der Sprachcode muss aus zwei Kleinbuchstaben bestehen. Buchstabenkombinationen, die keinem offiziell registriertem Code entsprechen, sind nicht explizit erlaubt, sollten bei der Verarbeitung seitens des Clienten aber nicht zu Fehlern führen.
   Ein Sprache–Text–Mapping darf keine zwei Einträge mit dem gleichen Sprachcode enthalten!

   Wenn der Parameter *pLanguages* eine leere Sequenz ist, enthält das Sprache–Text–Mapping alle in der Produktdatenbank hinterlegten sprach-spezifischen Beschreibungen.
   Anderenfalls enthält das Mapping die sprach-spezifischen Beschreibungen für die im Parameter angeforderten Sprachen. (Wenn für eine angeforderte Sprache keine Beschreibung für die Klasse hinterlegt ist, wird die ID der Klasse selber verwendet.)

Die Standardimplementierung delegiert an die gleichnamige Methode des globalen Produktdatenmanagers (Basistyp *OiPDManager*), wobei die implizite Instanz und die geforderten Sprachen als Argumente übergeben werden[14].

- *getPDInfo(pLanguage(String))* → *Any[] | Void* **new**

  Die Funktion liefert zusätzliche Produktinformationen zu dem Artikel, der durch die implizite Instanz repräsentiert wird.

  Der Parameter *pLanguage* gibt die Sprache an, die für textuelle Informationen verwendet werden soll.

[12] www.eclass.eu
[13] www.unspsc.org
[14] Die Standardimplementierung in *OiPDManager* wiederum delegiert an die gleichnamige Methode der zuständigen Produktdatenbank (Basistyp *OiProductDB*).

Rückgabewert ist ein Vektor von Informationselementen oder ein Wert vom Typ *Void*, wenn keine zusätzlichen Produktdateninformationen vorhanden sind.

Der Inhalt und die Reihenfolge der Elemente hängt von der zuständigen Produktdatenbank ab[15].

Die Standardimplementierung delegiert an die gleichnamige Methode des globalen Produktdatenmanagers (Basistyp *OiPDManager*), wobei die implizite Instanz und die gewünschte Sprache als Argumente übergeben werden[16].

Produktdatenbanken, die auf dem im OFML Part IV (OCD) standardisierten Format für kaufmännische Daten basieren [ocd], müssen mindestens die beiden folgenden Informationen liefern:

1. kaufmännisches Herstellerkürzel (*String*)
2. kaufmännisches Serienkürzel (*String*)

- *getOrderUnit(pLanguage(String | Void)) → String* **new**

  Die Funktion liefert die Bestelleinheit für den Artikel, der durch die implizite Instanz repräsentiert wird.

  Wenn der Parameter *pLanguage* ein Wert vom Typ *Void* ist, wird der sprach-unabhängige Code gemäß der UNECE Recommendation 20 geliefert [rec20], z.B. `C62` – *Stück*, `MTR` – *Meter* und `MTK` – *Quadratmeter*.

  Wenn der Parameter *pLanguage* ein Sprachcode (Typ *String*) ist, wird die Bezeichnung der Einheit in der geforderten Sprache geliefert.

  Enthalten die Produktdaten keine Informationen über die Bestelleinheit des Artikels, wird die Standardeinheit `C62` bzw. "Stück" angenommen und zurückgegeben.

  Die Standardimplementierung delegiert (zunächst) an die gleichnamige Methode des globalen Produktdatenmanagers (Basistyp *OiPDManager*), wobei die implizite Instanz als Argument übergeben wird.
  Wenn der Parameter *pLanguage kein* Wert vom Typ *Void* ist, wird anschliessend für die gebräuchlisten Einheiten mittels einer Text-Ressource für den sprach-unabhängigen Code die Bezeichnung in der geforderten Sprache ermittelt[17]. (Wenn keine Text–Ressource vorhanden ist, wird der sprach-unabhängige Code selber zurückgegeben.)

- *getArticleAttribute(pAttr(Symbol)) → Any* **new**

  Die Funktion liefert für den Artikel, der durch die implizite Instanz repräsentiert wird, den Wert für das angeforderte Attribut.

  Aktuell sind folgende Attribute definiert:

  `@Discountable`
  : Das Attribut vom Typ *Int* gibt an, ob auf den ggf. hinterlegten Einkaufspreis des Artikels Rabatte angewendet werden können oder nicht. Der Wert 0 (nein) bedeutet dabei, dass für den Artikel abweichend von der allgemeinen Konditionierung des Herstellers keine Abschläge vom Einkaufspreis erlaubt sind.

  Wenn ein nicht definiertes Attribut angefordert wird oder wenn die Produktdaten für den Artikel keine Informationen über das Attribut enthalten, wird ein Wert vom Typ *Void* zurückgegeben.

  Die Standardimplementierung delegiert an die gleichnamige Methode des globalen Produktdatenmanagers (Basistyp *OiPDManager*), wobei als erstes Argument das Objekt übergeben wird, welches durch die Methode *getArticleObj()* (s.o.) der impliziten Instanz geliefert wird, und als zweites Argument das angeforderte Attribut.

[15] Die zuständige Produktdatenbank ergibt sich aus der Programm-Zugehörigkeit der impliziten Instanz, s. 2.1.
[16] Die Standardimplementierung in *OiPDManager* wiederum delegiert an die gleichnamige Methode der zuständigen Produktdatenbank (Basistyp *OiProductDB*).
[17] siehe globale Funktion *oiGetOrderUnitDescr()*

Anmerkungen zu den Funktionen *getPDInfo()*, *getOrderUnit()* und *getArticleAttribute()*:

- Diese Funktionen wurden in dieser Reihenfolge mit zeitlichem Abstand eingeführt und dienen alle dem Abrufen spezifischer Information zu einem Artikel. Aus konzeptioneller Sicht wäre eine einheitliche, allgemeine Funktion wünschenswert, so dass nicht mit jeder neu hinzukommenden, relevanten Information eine neue Funktion eingeführt werden muss. Dem wurde mit der Einführung der Funktion *getArticleAttribute()* Rechnung getragen. Aus Gründen der Abwärtskompatibilität wurden dabei aber die beiden anderen Funktionen beibehalten.
- Für Clienten ist es aus Performanzgründen oft wünschenwert, mehrere Informationen verschiedenen Typs mit einem Funktionsaufruf abrufen zu können. Da sich die Bedürfnisse verschiedener Clienten bzgl. des Umfangs der abzurufenden Informationen unterscheiden, kann eine solche Funktion aber nicht allgemeingültig im Rahmen der Schnittstelle *Article* definiert werden. Dies muss bei Bedarf also in spezifischen OFML-Bibliotheken erfolgen.

2.5 Eigenschaften und Variantentext

- *getArticleFeatures(pLanguage(String | Void)* → *Any* **mod**

  Die Funktion liefert eine Beschreibung der aktuell definierten konfigurierbaren Merkmale (Eigenschaften) des Artikels, der durch die implizite Instanz repräsentiert wird.

  Die Funktion liefert entweder einen Wert vom Typ *Void*, falls keine Eigenschaftsbeschreibung für die implizite Instanz vorliegt, oder eine Liste von zweistelligen Vektoren:

  1. Merkmal (*String*)
  2. aktueller Wert des Merkmals (*String* | *Void*)

  Der Inhalt der Vektor-Elemente hängt vom Parameter *pLanguage* ab:

  – Hat der Parameter einen Wert vom Typ *Void*, werden sprachunabhängige Bezeichner für Merkmale und Werte geliefert. (Numerische Werte werden dazu in den Typ *String* konvertiert[18].)

    Diese Form des Funktionsaufrufes kann von OFML–Applikationen verwendet werden, um eine Beschreibung der Artikelkonfiguration zu erzeugen, die an ein externes Produktions- und Planungssystems (PPS) zur Auftragsabwicklung exportiert werden kann.

  – Enthält der Parameter einen Sprachcode (Typ *String*), werden die Bezeichnungen für Merkmale und Werte in der geforderten Sprache geliefert. Das Wert-Element kann dabei ein Wert vom Typ *Void* sein, siehe unten.

    Diese Form des Funktionsaufrufes wird von OFML–Applikationen zur Erzeugung des sogenannten **Variantentextes** verwendet. Dafür gelten folgende Regeln:

    * Jeder Vektor in der Rückgabeliste erzeugt eine Zeile des Variantentextes.
      (Die Zeichenketten in den Elementen für Merkmal und Wert dürfen somit selber keine Zeichen für einen Zeilenumbruch enthalten[19].)
    * Wenn das Wert-Element vom Typ *Void* ist, enthält das Merkmal-Element die komplette Zeile für den Variantentext.
      Das kann für mehrzeilige Beschreibungen von Merkmalen genutzt werden (die dann durch unmittelbar aufeinanderfolgende Einträge in der Rückgabeliste realisiert werden).
    * Wenn beide Vektor-Elemente eine Zeichenkette enthalten, ergibt sich die Zeile des Variantentextes aus der Verkettung beider Zeichenketten, getrennt durch die Zeichenkette ": ".

  In beiden Formen enthält die Rückgabeliste nur aktuell sichtbare/gültige konfigurierbare Merkmale mit einem definierten Wert. Eine Ausnahme sind nicht ausgewählte gültige optionale Merkmale mit einer speziellen Beschreibung für den nicht ausgewählten Zustand: diese sind in der Rückgabeliste enthalten, wenn der Parameter *pLanguage* einen Sprachcode enthält.

[18] siehe Konstruktoren des Typs *String*
[19] Falls doch vorhanden, werden sie von den Applikationen durch ein Leerzeichen ersetzt.

Andererseits können bestimmte definierte und sichtbare konfigurierbare Merkmale in der Rückgabeliste für den Variantentext ausgelassen werden. Dies kann der Fall sein, wenn das Merkmal indirekt durch abhängige Merkmale beschrieben wird, siehe z.B. den Text-Steuerungscode in der OCD–Merkmalstabelle [ocd].

Die Standardimplementierung delegiert an die gleichnamige Methode des globalen Produktdatenmanagers (Basistyp *OiPDManager*), wobei die implizite Instanz und die gewünschte Sprache als Argumente übergeben werden[20].

- *getArticleFeatures2(pLanguage(String)* → *Any* **new**

  Die Funktion liefert eine alternative Beschreibung der aktuell definierten konfigurierbaren Merkmale (Eigenschaften) des Artikels, der durch die implizite Instanz repräsentiert wird.

  Die Funktion liefert entweder einen Wert vom Typ *Void*, falls keine Eigenschaftsbeschreibung für die implizite Instanz vorliegt, oder eine Liste von Vektoren, die je ein Merkmal beschreiben:

  1. sprachunabhängiger Name des Merkmals (*String*)
  2. sprachspezifische Bezeichnung des Merkmals (*String*)
  3. aktueller Wert des Merkmals (*String* | *Void*)[21]
  4. sprachspezifische Bezeichnung Wertes (*String*)

  Die sprachspezifischen Elemente werden in der Sprache geliefert, die durch den Parameter *pLanguage* spezifiziert ist.

  Die Rückgabeliste enthält nur aktuell sichtbare/gültige konfigurierbare Merkmale mit einem definierten Wert. Eine Ausnahme sind nicht ausgewählte gültige optionale Merkmale mit einer speziellen Beschreibung für den nicht ausgewählten Zustand: diese sind in der Rückgabeliste mit einem Wert vom Typ *Void* im 3. Element enthalten.

  Im Gegensatz zur Funktion *getArticleFeatures()* (s.o.) wird für Merkmalswerte mit einer mehrzeiligen Beschreibung (Variantentext) im 4. Element nur die erste Zeile verwendet/zurückgegeben (s.a. den Text-Steuerungscode in der OCD–Merkmalstabelle [ocd]).

  Die Standardimplementierung delegiert an die gleichnamige Methode des globalen Produktdatenmanagers (Basistyp *OiPDManager*), wobei die implizite Instanz und die gewünschte Sprache als Argumente übergeben werden.

- *getAllArticleFeatures(pLanguage(String)* → *Any* **new**

  Die Funktion liefert eine Beschreibung aller aktuellen Merkmalsbelegungen (Eigenschaften) des Artikels, der durch die implizite Instanz repräsentiert wird.

  Im Gegensatz zu den Funktionen *getArticleFeatures()* und *getArticleFeatures2()* (s.o.) kann dies interne Merkmale umfassen, die für den Anwender nicht sichtbar/konfigurierbar sind.

  Die Funktion liefert entweder einen Wert vom Typ *Void*, falls keine Eigenschaftsbeschreibung für die implizite Instanz vorliegt, oder eine Liste von Vektoren, die je ein Merkmal beschreiben:

  1. sprachunabhängiger Name des Merkmals (*String*)
  2. sprachspezifische Bezeichnung des Merkmals (*String*)
  3. aktueller Wert des Merkmals (*String* | *Void*)[22]
  4. sprachspezifische Bezeichnung Wertes (*String*)
  5. Kennzeichen (*Int*), das angibt, ob das Merkmal sichtbar/konfigurierbar ist (1) oder nicht (0).

  Die sprachspezifischen Elemente werden in der Sprache geliefert, die durch den Parameter *pLanguage* spezifiziert ist.

[20] Die Standardimplementierung in *OiPDManager* wiederum delegiert an die Methode *getPropDescription()* der zuständigen Produktdatenbank (Basistyp *OiProductDB*).
[21] Numerische Werte werden in den Typ *String* konvertiert.
[22] Numerische Werte werden in den Typ *String* konvertiert.

Die Übermittlung interner Merkmale ist optional, d.h. sie hängt von der Implementierung und dem zugrundeliegenden Datenformat der zuständigen Produktdatenbank ab[23].

Wenn in den Produktdaten für interne (nicht sichtbare) Merkmale keine sprachspezifische Bezeichnung für das Merkmal bzw. den Wert hinterlegt ist, enthält das 2. Element eine leere Zeichenkette bzw. ist das 4. Element identisch mit dem 3. Element.

Die Rückgabeliste enthält nur Merkmale, die aktuell einen definierten Wert haben. Eine Ausnahme sind nicht ausgewählte gültige und konfigurierbare optionale Merkmale mit einer speziellen Beschreibung für den nicht ausgewählten Zustand: diese sind in der Rückgabeliste mit einem Wert vom Typ *Void* im 3. Element enthalten.

Die Standardimplementierung delegiert an die gleichnamige Methode des globalen Produktdatenmanagers (Basistyp *OiPDManager*), wobei die implizite Instanz und die gewünschte Sprache als Argumente übergeben werden.

- *getArticleFeaturesDescr(pType(Symbol), pLanguage(String) → Any* **new**

  Die Funktion liefert eine Beschreibung der Merkmale (Eigenschaften) des Artikels, der durch die implizite Instanz repräsentiert wird.

  Die gewünschte Form der Beschreibung wird im Parameter *pType* angegeben.

  Die sprachspezifischen Elemente werden in der Sprache geliefert, die durch den Parameter *pLanguage* spezifiziert ist.

  Aktuell sind folgende Beschreibungstypen definiert:

  `@Text`
  : Die Beschreibung entspricht dem Ergebnis des Funktionsaufrufes `getArticleFeatures(pLanguage)`

  `@AllIDs`
  : Die Beschreibung entspricht dem Ergebnis des Funktionsaufrufes `getAllArticleFeatures(pLanguage)`

  `@ID_Text`
  : Die Funktion liefert entweder einen Wert vom Typ *Void*, falls keine Eigenschaftsbeschreibung für die implizite Instanz vorliegt, oder eine Liste von Vektoren, die je ein aktuell definiertes konfigurierbares Merkmal beschreiben:
    1. sprachunabhängiger Name des Merkmals (*String*)
    2. aktueller Wert des Merkmals (*String* | *Void*)[24]
    3. Liste von *String*–Paaren (Typ *Vector*), die je eine Zeile der sprachspezifischen Beschreibung des Merkmals spezifizieren:
       Die erste Zeichenkette ist für die sprachspezifische Bezeichnung des Merkmals vorgesehen und die zweite für die sprachspezifische Bezeichnung Wertes. Für die Bildung der zugehörigen Zeile der Merkmalsbeschreibung seitens des Clienten gelten die Bestimmungen für die Bildung des Variantentextes, die bei der Funktion *getArticleFeatures()* genannt sind.

    Die Rückgabeliste enthält nur aktuell sichtbare/gültige konfigurierbare Merkmale mit einem definierten Wert. Eine Ausnahme sind nicht ausgewählte gültige optionale Merkmale mit einer speziellen Beschreibung für den nicht ausgewählten Zustand: diese sind in der Rückgabeliste mit einem Wert vom Typ *Void* im 2. Element enthalten.

    Andererseits können bestimmte definierte und sichtbare konfigurierbare Merkmale in der Rückgabeliste ausgelassen werden. Siehe dazu auch die entsprechende Anmerkuung bei der Funktion *getArticleFeatures()*.

  Die Standardimplementierung ruft beim Beschreibungstyp `@Text` die Funktion *getArticleFeatures()* und beim Beschreibungstyp `@AllIDs` die Funktion *getAllArticleFeatures()*. Bei allen anderen Beschreibungstypen wird an die gleichnamige Methode des globalen Produktdatenmanagers (Basistyp

[23] Die zuständige Produktdatenbank ergibt sich aus der Programm-Zugehörigkeit der impliziten Instanz, s. 2.1.
[24] Numerische Werte werden in den Typ *String* konvertiert.

OiPDManager) delegiert, wobei die implizite Instanz, der geforderte Beschreibungstyp und die gewünschte Sprache als Argumente übergeben werden.

2.6 Persistenz und Aktualisierung

Prinzipiell können und müssen folgende zwei Formen einer persistenten Repräsentation eines Artikels betrachtet werden:

Dump–Repräsentation
: Bei dieser Form wird der komplette interne Zustand (Membervariablen) der OFML–Instanz selber serialisiert, die einen Artikel repräsentiert.

  Diese Form kann von Applikationen genutzt werden, um eine komplette OFML–Szene zu speichern oder um den Zustand der OFML–Instanz in das Clipboard zu schreiben bzw. die Instanz aus dem im Clipboard gespeicherten Zustand wieder herzustellen.

  Beim Speichern und Laden eines Dumps kommen die in [ofml] spezifizierten Persistenzregeln zur Anwendung.

  Das Dump–Format ist nicht standardisiert[25].

Warenkorb–Repräsentation
: Bei dieser Form werden alle relevanten Informationen zu dem Artikel gespeichert, die notwendig sind, um diesen in Bestellprozessen (weiter) verarbeiten zu können. Dies umfasst auch Informationen, die benötigt werden, um die OFML–Instanz wieder zu erzeugen, z.B. wenn der Artikel umkonfiguriert werden soll.

  Für die Wiederherstellung der OFML–Instanz aus einer Warenkorb–Repräsentation werden folgende Informationen benötigt:

  - die Programm-Zugehörigkeit, siehe Funktion *getProgram()* (Abschn. 2.1)
  - die (Grund-)Artikelnummer, siehe Funktion *getArticleSpec()* (Abschn. 2.2)
  - der OFML–Variantencode, siehe Funktion *getXArticleSpec()* (Abschn. 2.2)
  - der sogenannte *AddStateCode*, siehe Funktion *getAddStateCode()* (Abschn. 2.3)

  Das Warenkorb–Format ist nicht standardisiert[26].

Die in diesem Abschnitt unten spezifizierten Funktionen dienen zur Realisierung eines allgemeingültigen Konzepts für die Behandlung von bereits erzeugten und gespeicherten Artikeln in Bezug auf die aktuell installierten Produktdaten. Diese können von den Produktdaten abweichen, mit denen der Artikel zuletzt erzeugt bzw. umkonfiguriert und gespeichert wurde.

Die grundlegenden Festlegungen dieses Konzepts sind:

- Unmittelbar nach dem Laden einer gespeicherten Planung bzw. eines gespeicherten Warenkorbes besitzen die darin enthaltenen Artikel alle Eigenschaften (Konfiguration, Texte, Preis) wie zum Zeitpunkt der Speicherung.

  Der Zustand der Artikel in Bezug auf die aktuell installierten Produktdaten ist zunächst *unbekannt/undefiniert* (`Undefined`), da in der Regel die Version der beim Speichern verwendeten Produktdaten nicht bekannt ist und davon ausgegangen werden muss, dass sich die Produktdaten in der Zwischenzeit geändert haben.

  Sieht das verwendete Warenkorb–Format die Speicherung der verwendeten Version der Produktdaten vor, kann die Applikation nach dem Abgleich mit der Version der aktuell installierten Produktdaten den Status der betreffenden Artikel ggf. auch auf den Zustand *aktuell/aktualisiert* (`Up2Date`) setzen[27]. Für gespeicherte OFML–Szenen (Dump–Format) besteht diese Möglichkeit nicht, da die Schittstelle *Article* (aktuell) keine Zuweisung der Produktdaten-Version an eine Artikelinstanz vorsieht.

[25] In den Anwendungen von EasternGraphics wird das FML–Format verwendet.
[26] In den Anwendungen von EasternGraphics wird das OBX–Format verwendet.
[27] Das in den Anwendungen von EasternGraphics verwendete OBX–Format sieht diese Möglichkeit nicht vor.

- Um den Artikel bearbeiten (umkonfigurieren) zu können, oder um Texte und Preise aus den aktuell installierten Produktdaten zu verwenden, muss der Artikel *aktualisiert* werden.

  Nach der Aktualisierung befindet sich der Artikel im Zustand *aktuell/aktualisiert* (`Up2Date`).

  Solange ein Artikel nicht diesen Zustand besitzt, kann/darf er nicht verändert werden (Angebotsbindung)!

- Bevor ein Artikel aktualisiert wird, muss festgestellt werden, ob der Artikel überhaupt aktualisierbar ist. Dazu muss er folgende Bedingungen erfüllen:

  1. Der Artikel ist in den aktuell installierten Produktdaten in derselben kaufmännischen Serie enthalten.
  2. Die gespeicherte Konfiguration des Artikels kann mit den aktuell installierten Produktdaten wieder hergestellt werden, d.h., alle laut der gespeicherten Konfiguration durch den Anwender veränderbaren (konfigurierbaren) Merkmale sind auch in den aktuellen Produktdaten konfigurierbar und die gespeicherten Bewertungen dieser Merkmale können auch mit den aktuellen Produktdaten angenommen werden[28].

  Als Basis für die Überprüfung der Aktualisierbarkeit wird der hersteller-neutrale OFML–Variantencode empfohlen, siehe Abschn. 2.2 und Anhang C.

  Nach der Überprüfung der Aktualisierbarkeit kann sich der Artikel in 3 Zuständen befinden:
  - *ungültig* (`Invalid`), wenn Bedingung 1 nicht erfüllt ist
  - *aktualisierbar* (`Updatable`), wenn beide Bedingungen erfüllt sind
  - *migrierbar* (`Migratable`), wenn Bedingung 1 erfüllt ist, aber nicht Bedingung 2

  Die Überprüfung der Aktualisierbarkeit stellt konzeptionell einen eigenständigen Schritt dar, unabhängig von der eigentlichen Aktualisierung[29]. Aus Anwendersicht werden beide Schritte in der Regel jedoch durch die Applikationen zu einer Aktion zusammengefasst.

- Die *Migration* ist eine besondere Form der Aktualisierung, bei der in Kauf genommen wird, dass der gespeicherte Konfigurationszustand nicht exakt gleich wieder hergestellt werden kann. Bei der Migration wird versucht, möglichst viele der gespeicherten Merkmalsbewertungen zu übernehmen.

  Es ist den OFML–Applikationen überlassen, ob sie die Migration unterstützen, und falls ja, ob und in welcher Form der Anwender dabei einbezogen wird.

Anhang D enthält eine grafische Darstellung der möglichen Zustände und ihrer Zusammenhänge.

Die oben beschriebenen Grundprinzipien werden durch folgende spezifischen Bestimmungen ergänzt:

- *Komposite Artikel* dürfen nur in ihrer Gesamtheit aktualisiert werden, d.h., sobald einer der Artikel des Verbundes nicht aktualisiert werden kann, dürfen auch alle anderen Artikel der Zusammensetzung nicht aktualisiert werden.

- Eine Artikelisntanz, die per *Copy/Cut und Paste* eingefügt wurde, übernimmt den Update-Zustand der Original-Instanz.

Die folgenden Funktionen dienen zur Realisierung des oben beschriebenen Konzepts:

- *setupConfiguration(pBaseArticle(String), pArticleCode(String), pCodeType(Symbol), pMigration(Int)) → Int* **new**

  Die Funktion weist der impliziten Artikelinstanz die übergebene Grundartikelnummer zu und versucht dann, den durch den übergebenen Article–Code repräsentierten Konfigurationszustand auf der Basis der aktuell installierten Produktdaten (wieder) herzustellen.

[28] Diese Bedingung ist also **nicht** erfüllt, wenn ein konfigurierbares Merkmal in den aktuellen Produktdaten nicht (mehr) enthalten oder nicht (mehr) konfigurierbar ist, oder wenn ein gespeicherter Wert eines konfigurierbaren Merkmals in den aktuellen Produktdaten nicht (mehr) gültig ist. Ein in den aktuellen Produktdaten hinzugekommenes konfigurierbares Merkmal hingegen verletzt die Bedingung nicht.

[29] Das spiegelt sich auch in den in diesem Abschnitt spezifizierten Funktionen wieder.

Die zu verwendende Produktdatenbank ergibt sich aus der Programm-Zugehörigkeit der impliziten Instanz, s. Abschn. 2.1.

Der Parameter *pCodeType* dient zur Angabe des Typs des übergebenen Article–Codes. Aktuell sind die Typen `@VarCode` und `@OFMLVarCode` erlaubt, siehe Funktion *setXArticleSpec()* im Abschn. 2.2. Laut der Empfehlung im oben beschriebenen Konzept sollte vorzugweise der Hersteller-neutrale OFML–Variantencode verwendet werden[30].

Diese Funktion ist primär für die Wiederherstellung einer Artikelinstanz anhand einer gespeicherten Warenkorb–Repräsentation vorgesehen.

Falls keine Produktdatenbank für das OFML–Programm der impliziten Instanz installiert ist, oder wenn diese keinen Artikel mit der angegebenen Grundartikelnummer enthält, bleibt der Konfigurationszustand der impliziten Instanz unverändert[31], ihr Update-Zustand wird auf `@Invalid` gesetzt und der Rückgabewert ist 0.

Kann der Konfigurationszustand vollständig hergestellt werden, wird der Update-Zustand der impliziten Instanz auf `@Up2Date` gesetzt und der Rückgabewert ist 1.

Kann der Konfigurationszustand *nicht* vollständig hergestellt werden und hat der Parameter *pMigration* den Wert 0 (false), bleibt der Konfigurationszustand der impliziten Instanz unverändert[32], ihr Update-Zustand wird auf `@Migratable` gesetzt, und der Rückgabewert ist 0.

Kann der Konfigurationszustand *nicht* vollständig hergestellt werden und hat der Parameter *pMigration* den Wert 1 (true), so nimmt die Artikelinstanz den teilweise hergestellten Konfigurationszustand an, ihr Update-Zustand wird auf `@Up2Date` gesetzt, und der Rückgabewert ist 0.

Im Fall einer Aktualisierung bzw. Migration erfolgt die Zuweisung der Grundartikelnummer und des Variantencodes mittels unmittelbar aufeinander folgender Aufrufe der Methoden *setArticleSpec()* und *setXArticleSpec()* auf der impliziten Instanz (s. Abschn. 2.2).

Die Standardimplementierung delegiert an die gleichnamige Methode des globalen Produktdatenmanagers (Typ *OiPDManager*).

- *updateConfiguration()* → *Void* **new**

  Die Funktion aktualisiert bzw. migriert den Konfigurationszustand der impliziten Artikelinstanz auf der Basis der aktuell installierten Produktdaten.

  Die zu verwendende Produktdatenbank ergibt sich aus der Programm-Zugehörigkeit der impliziten Instanz, s. Abschn. 2.1.

  Diese Funktion ist primär für die Aktualisierung bzw. Migration einer Artikelinstanz vorgesehen, die aus einer Dump–Repräsentation geladen wurde.

  Falls die implizite Instanz einen Unterartikel eines kompositen Artikels repräsentiert, so delegiert die Standardimplementierung den Request an die in der Objekthierarchie über der impliziten Instanz auf dem höchsten Level befindliche komposite Artikelinstanz.

  Anderenfalls verhält sich die Standardimplementierung wie folgt:

  1. Die Funktion hat keinen Effekt, wenn der Update-Zustand `@Up2Date` oder `@Invalid` ist.
  2. Falls der aktuelle Update-Zustand `@Undefined` ist, wird zunächst die Methode *checkObjUpdatability()* des globalen Produktdatenmanagers (Basistyp *OiPDManager*) gerufen, wobei die implizite Instanz und `@OFMLVarCode` als Codierungstyp übergeben werden. Der von dieser Methode zurückgegebene Update-Zustand wird dann an die implizite Instanz zugewiesen, siehe Funktion *setObjState()* (Abschn. 2.3).
  3. Falls der aktuelle Update-Zustand der impliziten Instanz nun `@Invalid` ist, oder `@Migratable` und der Anwender keine Migration durchführen möchte, wird die Funktion beendet.
  4. Jetzt erfolgt die kaufmännische Aktualisierung (`@Updatable`) bzw. Migration (`@Migratable`) durch Delegation an die gleichnamige Methode des globalen Produktdatenmanagers (Basistyp *OiPDManager*), wobei die implizite Instanz als Argument übergeben wird.

[30] was für die Applikationen von EasternGraphics zutrifft
[31] Auch die übergebene Grundartikelnummer wird dann nicht übernommen.
[32] Auch die übergebene Grundartikelnummer wird dann nicht übernommen.

5. Nun erfolgt der Aufruf der Methode *updateGeometry()* (s.u.) auf der impliziten Instanz, um ggf. Anpassungen an der Geometrie vorzunehmen.
6. Abschließend wird der Update-Zustand der impliziten Instanz auf `@Up2Date` gesetzt, siehe Funktion *setObjState()* (Abschn. 2.3).

- *checkUpdatability(pCodeType(Symbol))* → *Symbol* **new**

  Die Funktion prüft, ob in den aktuell installierten Produktdaten der Artikel mit der Grundartikelnummer der impliziten Instanz enthalten ist, und ob mit diesen Produktdaten der Konfigurationszustand vollständig wieder hergestellt werden kann, der im Article-Code der impliziten Instanz codiert ist.

  Die zu verwendende Produktdatenbank ergibt sich aus der Programm-Zugehörigkeit der impliziten Instanz, s. Abschn. 2.1.

  Der Parameter *pCodeType* bestimmt den Typ des zu verwendenden Article-Codes. Aktuell sind die Typen `@VarCode` und `@OFMLVarCode` zugelassen, siehe Funktion *setXArticleSpec()* (Abschn. 2.2).

  Der Rückgabewert ist `@Invalid`, wenn keine Produktdatenbank für das OFML–Programm der impliziten Instanz installiert ist oder wenn diese keinen Artikel mit der Grundartikelnummer der impliziten Instanz enthält.

  Der Rückgabewert ist `@Migratable`, wenn die Produktdatenbank den Artikel mit der Grundartikelnummer der impliziten Instanz enthält, der durch den Article-Code des spezifizierten Typs beschriebene Konfigurationszustand aber nicht vollständig wieder hergestellt werden kann.

  Der Rückgabewert ist `@Updatable`, wenn die Produktdatenbank den Artikel mit der Grundartikelnummer der impliziten Instanz enthält und der durch den Article-Code des spezifizierten Typs beschriebene Konfigurationszustand vollständig wieder hergestellt werden kann.

  Der Methodenaufruf verändert den Update-Zustand der impliziten Instanz selber nicht.

  Die Standardimplementierung delegiert an die Methode *checkObjUpdatability()* des globalen Produktdatenmanagers (Basistyp *OiPDManager*), wobei die implizite Instanz und der spezifizierte Codierungstyp übergeben werden.

- *updateGeometry()* → *Void* **new**

  Die Funktion aktualisiert die Geometrie der impliziten Instanz entsprechend ihrer aktuellen Artikelkonfiguration.

  Die Funktion wird von der Standardimplementierung der Funktion *updateConfiguration()* (s.o.) aufgerufen.

  Die Standardimplementierung ist leer.

  Die Implementierung im Basistyp *OiOdbPlElement* aktualisiert die ODB–Objekthierarchie entsprechend dem aktuellen Zustand der Hash-Tabelle der ODB–Parameter (welche sich nach der vorangegangenen kaufmännischen Aktualisierung bzw. Migration geändert haben können).

2.7 Konsistenzprüfung und Preisdatum

- *checkConsistency()* → *Int* | *Void* **mod**

  Die Funktion prüft die Konsistenz und Vollständigkeit der impliziten Instanz.

  Gegebenenfalls werden Korrekturen oder Ergänzungen vorgenommen bzw. Fehlermeldungen generiert.

  Wenn die Konsistenz und Vollständigkeit der Instanz nicht eindeutig bestimmt werden kann, wird ein Wert vom Typ *Void* zurückgegeben. Anderenfalls ist der Rückgabewert 1 (ok), wenn die Instanz konsistent und vollständig ist, und 0, wenn nicht.

Wurde von der übergeordneten Instanz, die die Konsistenzprüfung der impliziten Instanz veranlaßt hat, ein *Fehler-Log* angelegt, so sind die Fehlermeldungen in dieses Fehler-Log zu schreiben, anderenfalls können sie mittels der globalen Funktion *oiOutput()* direkt an den Anwender ausgegeben werden.

Das zu verwendende Fehler-Log muß mittels Methode *getErrorLog()* von der globalen Planungsinstanz (Typ *OiPlanning*) abgerufen werden.

Die für *checkConsistency()* festgelegte Datenstruktur des Fehler-Logs ist eine Hash-Tabelle, in der für jede Artikelinstanz die betreffenden Meldungen eingetragen sind. Als Schlüssel für die Hash-Tabelle wird entweder de Bestell-ID der Artikelinstanz (s. Funktion *getOrderID()* im Abschn. 2.8 unten) oder ihr absoluter hierarchischer Objektname (s. Funktion *getName()* der Schnittstelle *MObject*) verwendet.

Der Wert zu diesem Schlüssel ist eine Liste von dreistelligen Vektoren:

1. die Fehlermeldung (String)
2. der absolute hierarchische Objektname des Instanz, die den Fehler gemeldet hat (String)
3. der Name der Methode, in der der Fehler festgestellt wurde (String)

Das Verhalten der Standardimplementierung ist wie folgt:

- Wenn in der globalen Planungsinstanz (Typ *OiPlanning*) laut Methode *getErrorLog()* eine Hash-Tabelle für das Fehler-Log angelegt ist, werden Inkonsistenzmeldungen in dieser Hash-Tabelle abgelegt, anderenfalls erfolgt eine Ausgabe der Meldungen an den Anwender mittels globaler Funktion *oiOutput()*.
- Ist der Update-State der impliziten Instanz *nicht* `@Up2Date` (s. Abschn. 2.6 und Anh. D), wird als Ergbnis der Rückgabewert der Methode *getObjState()* der impliziten Instanz für den Zustandstyp `@OI_ConsistencyState` (s. Abschn. 2.3) verwendet und die Funktion beendet.
- Die kaufmännische Konsistenzprüfung erfolgt durch Delegation an die gleichnamige Methode des globalen Produktdatenmanagers (Basistyp *OiPDManager*), wobei die implizite Instanz als Argument übergeben wird.

  Der Umfang der kaufmännischen Konsistenzprüfung hängt von der Implementierung und dem zugrundeliegenden Datenformat der zuständigen Produktdatenbank ab[33].

  Unterstützt die zuständige Produktdatenbank bei der Preisermittlung Gültigkeitszeiträume für Preiskomponenten, ist gefordert, dass geprüft wird, ob in den (aktuell installierten) Produktdaten zu dem Preisdatum der impliziten Instanz (s. Methode *getPriceDate()* unten) Preisinformationen hinterlegt sind.
- Nach der kaufmännischen Konsistenzprüfung erfolgt eine Delegation an die Methode *checkObjConsistency()* der Instanz von *OiProgInfo*, welche in der globalen Planungsinstanz (Basistyp *OiPlanning*) für das OFML–Programm der impliziten Instanz registriert ist. Dabei wird die implizite Instanz als Argument übergeben.

  Dies kann zur Durchführung zusätzlicher, programm-spezifischer Prüfungen für die implizite Instanz genutzt werden.
- Das Ergebnis der Konsistenzprüfung wird in der impliziten Instanz mittels Aufruf der Methode *setObjState()* für den Zustandstyp `@OI_ConsistencyState` gespeichert (s. Abschn. 2.3).

- *setPriceDate(pDate(String))* → *Void* **new**

  Die Funktion weist der impliziten Instanz das Datum zu, welches während der Preisermittlung zu verwenden ist.

  Das Datum muss im Format `YYYYMMDD` angegeben werden. Wann das Datum nicht in diesem Format übergeben wird, hat die Funktion keinen Effekt.

[33] Die zuständige Produktdatenbank ergibt sich aus der Programm-Zugehörigkeit der impliziten Instanz, s. 2.1.

Wenn die implizite Instanz der Schnittstellen-Kategorie `@IF_Article` angehört, delegiert die Standardimplementierung an die gleichnamige Methode der Instanz, die durch die Funktion *getArticleObj()* geliefert wird (s. Abschn. 2.4). Anderenfalls weist die Standardimplementierung das übergebene Datum (wenn gültig) einer Membervariablen der impliziten Instanz zu.

- *getPriceDate()* → *String* | *Void* **new**

  Die Funktion liefert das Datum, welches während der Preisermittlung für die implizite Instanz zu verwenden ist.

  Wenn der impliziten Instanz kein (gültiges) Datum mittels der Methode *setPriceDate()* zugewiesen wurde (s.o.), wird ein Wert vom Typ *Void* zurückgegeben. Anderenfalls ist der Rückgabewert eine Zeichenkette im Format `YYYYMMDD`.

  Wenn die implizite Instanz der Schnittstellen-Kategorie `@IF_Article` angehört, delegiert die Standardimplementierung an die gleichnamige Methode der Instanz, die durch die Funktion *getArticleObj()* geliefert wird (s. Abschn. 2.4). Anderenfalls liefert die Standardimplementierung das Datum, das mittels vorangegangenem Aufruf der Methode *setPriceDate()* zugewiesen wurde bzw. einen Wert vom Typ *Void*, wenn zuvor kein (gültiges) Datum zugwiesen wurde.

2.8 Sonstige

- *getArticleObj()* → *MObject* **new**

  Die Funktion gibt das Objekt zurück, dessen Methoden der Schnittstelle *Article* verwendet werden müssen, um Informationen (Artikelnummern, Texte, Preis) über den Artikel zu erhalten, der durch die implizite Instanz repräsentiert wird.

  Die Standardimplementierung gibt die implizite Instanz selber zurück.

  Die Funktion kann bzw. muss in Metatypen überschrieben werden, um an ein spezielles (gekapseltes) Kindobjekt zu delegieren. Der Typ dieses Objektes muss die Schnittstelle *Article* implementieren!

  Clienten müssen diese Funktion rufen und die entsprechenden Funktionen der Schnittstelle auf dem zurückgegebenen Objekt aufrufen, um die gewünschten Artikelinformationen für die implizite Instanz zu erhalten!

- *getPDLanguage()* → *String* **new**

  Die Methode liefert die Sprache, welche für kaufmännische Texte und Property-bezogene Texte bei der impliziten Instanz verwendet werden soll.

  Die Standardimplementierung[34] basiert auf einer Membervariablen, die zu folgenden Zeitpunkten mittels eines Aufrufs der Methode *getPDLanguage()* auf der globalen Planungsinstanz (Basistyp *OiPlanning*)[35] initialisiert bzw. aktualisiert wird (wobei die implizite Instanz als Argument übergeben wird):

  - bei Erzeugung der Instanz
  - bei jeder Änderung der Programm-Zugehörigkeit der impliziten Instanz (s. Abschn. 2.1) via Aufruf der Methode *setProgram()* (nur bei Instanzen vom Typ *OiPlElement*)
  - bei jedem Aufruf der Methode *updateProperties()* der Schnittstelle *Property* [property] auf der impliziten Instanz, wenn deren Aktualisierungsstatus *@Up2Date* ist (s. Abschn. 2.6)

[34] im Basistyp *OiPart* ab OI Version 1.42.0 vom Herbst 2022
[35] deren Implementierung wiederum auf dem Application Callback *::ofml::app::getPDLanguage()* basiert

- *setOrderID(pID(Symbol)) → Void*

  Die Funktion weist der impliziten Instanz eine eindeutige Bestell–ID zu.

  Die Bestell–ID kann für die Synchronisation einer von der Applikation verwalteten Warenkorb–Struktur mit den Artikelinstanzen in der Planungshierarchie (OFML–Szene) verwendet werden. Die Bestell–ID dient dann zur Zuordnung eines Artikel–Postens im Warenkorb zu der Instanz, die den Artikel in der Planung repräsentiert.

  Die Bestell–ID wird der Artikelinstanz von der globalen Planungsinstanz (Basistyp *OiPlanning*) unmittelbar nach deren Erzeugung zugewiesen und während der Dauer der Existenz der Artikelinstanz nicht geändert.

  Ändert sich die Lage des Artikelinstanz in der Planungshierarchie (z.B. bei Gruppierungsaktionen) wird die Bestell–ID bei der dabei stattfindenden Cut/Paste-Operation von der zerstörten Instanz auf die neu erzeugte Klon-Instanz übertragen.

  Die Standardimplementierung speichert die übergebene ID in einer entsprechenden Membervariablen.

- *getOrderID() → Symbol | Void*

  Die Funktion liefert die eindeutige Bestell–ID der impliziten Instanz.

  Wenn der impliziten Instanz keine Bestell–ID zugewiesen wurde (s. Funktion *setOrderID()* oben), wird ein Wert vom Typ *Void* zurückgegeben. (Das trifft für Applikationen zu, die keine OFML–Szene verwalten.)

  Die Standardimplementierung liefert den aktuellen Wert der Membervariablen, an die die Zuweisung der ID während *setOrderID()* erfolgt.

- *setCatalogInfo(pInfo(Any[])) → Void* **new**

  Die Funktion weist der impliziten Instanz Informationen über den Eintrag im Katalog der Anwendung zu, der beim Einfügen/Erzeugen der Artikelinstanz verwendet wurde.

  Die Information wird als Liste von zweistelligen Vectoren übergeben:

  1. Infotyp (*Symbol*)
  2. Info (*Any*)

  Folgende Informationstypen sind definiert:

  `@CatID` ID des OFML–Programms, das die Katalogdaten enthält (*Symbol*).

  Diese ID kann von der Programm–ID der impliziten Instanz abweichen, wenn das OFML–Programm der Artikelinstanz keine Katalogdaten beinhaltet.

  `@CatV` Die Version des OFML–Pakets des OFML–Programms mit den Katalogdaten (*String*).

  `@ArtNr` Die im Katalogeintrag angegebene Grundartikelnummer (*String*).

  Diese kann von der Grundartikelnummer der impliziten Instanz[36] abweichen, wenn in Folge der Zuweisung des Variantencodes aus dem Katalogeintrag (s.u.) ein Artikeltausch stattfand.

  `@VarCode` Der im Katalogeintrag angegebene Variantencode zur Bewertung von Merkmalen abweichend von der initialen (Grund-)Konfiguration des Artikels (*String*).

  Welche dieser möglichen Informationseinheiten und in welcher Reihenfolge diese im Parameter *pInfo* übergeben werden, hängt von den Applikationen ab.

  Die Zuweisung erfolgt unmittelbar nach der kaufmännischen Initialsierung der Artikelinstanz (s.a. Anh. E).

  Die Standardimplementierung speichert die übergebene Information in einer entsprechenden Membervariablen.

[36] siehe Funktion *getArticleSpec()* in Abschn. 2.2)

- *getCatalogInfo()* → *Any[]* | *Void* **new**

  Die Funktion liefert Informationen über den Eintrag im Katalog der Anwendung, der beim Einfügen/Erzeugen der Artikelinstanz verwendet wurde.

  Wenn der impliziten Instanz keine Informationen zugewiesen wurden (s. Funktion *setCatalogInfo()* oben), wird ein Wert vom Typ *Void* zurückgegeben. Anderenfalls entspricht der Rückgabewert der Spezifikation für den Parameter *pInfo* der Funktion *setCatalogInfo()*.

  Die Information kann von der Applikation genutzt werden, um den Katalogeintrag zu identifizieren, der beim Einfügen/Erzeugen der Artikelinstanz verwendet wurde[37].

  Die Standardimplementierung liefert den aktuellen Wert der Membervariablen, an die die Information während *setCatalogInfo()* zugewiesen wird.

[37] z.B. um an Bilder zu kommen, die im Katalog zu dem Artikel hinterlegt sind

3 Die Schnittstelle *CompositeArticle*

3.1 Synchronisation mit der Warenkorb-Struktur

- *getSubArticleIDs()* → *String[]*

  Die Funktion gibt eine Liste oder einen Vektor mit den IDs für diejenigen Kind-Objekte der impliziten Instanz zurück, die in der Warenkorb-Struktur dargestellt werden müssen, d.h., die Unterartikel repräsentieren.

  Die IDs werden von der kompositen Instanz erstellt und verwaltet. Eine ID muss einen Unterartikel im Kontext der kompositen Instanz eindeutig identifizieren.

  Die ID eines Unterartikels darf sich während der Verarbeitung einer Änderung eines Merkmals des kompositen Artikels nicht ändern!

  Mögliche Verfahren für die Generierung der IDs sind:
  - Verwendung des OFML–Typs und/oder der Position des Kind-Objektes
  - Verwendung des lokalen Namens des Kind-Objektes
  - Verwendung des Merkmals der kompositen Instanz und dessen Wertes, welche die Erzeugung des Kind-Objektes bewirkt haben[38].

- *getSubArticle(pID(String))* → *MObject* | *Void*

  Die Funktion liefert die Referenz auf das Kind-Objekt der impliziten Instanz, das den Unterartikel mit der angegebenen ID repräsentiert.

  Es wird erwartet, dass die übergebene ID von einem vorherigen Aufruf der Methode *getSubArticleIDs()* der impliziten Instanz stammt (s.o.).

  Existiert kein Kind-Objekt mit der übergebenen ID, wird ein Wert vom Typ *Void* zurückgegeben.

Beim Erstellen der (persistenten) Warenkorb-Repräsentation werden für alle Unterartikel des kompositen Artikels entsprechende Unterpositionen in der Warenkorb-Struktur angelegt. Bei den Unterpositionen wird die jeweilige ID (aus *getSubArticleIDs()*) abgespeichert.

Damit kann dann die komposite Artikelinstanz wieder komplett aus der Warenkorb-Repräsentation hergestellt werden (s.a. Anh. E).

[38] Dabei müssen Situationen berücksichtigt werden, in denen eine bestimmte Merkmalseinstellung zur Erzeugung von mehr als einem Unterartikel führen kann.

A Alphabetischer Index der Funktionen

B Standard–Kategorien

Folgende Standard–Kategorien sind für Artikelinstanzen vordefiniert:

`@PseudoArticle`

Die Instanz repräsentiert einen Pseudo-Artikel, d.h., ist ein Objekt, das keinen physisch/real existierenden Artikel repräsentiert, sondern lediglich aus technischen (und ggf. historischen) Gründen die Schnittstelle *Article* implementiert. Solche Instanzen müssen somit bei der Bestellabwicklung in spezieller Weise behandelt werden.

Gehört eine Artikelinstanz dieser Kategorie an, wird in der persistenten Warenkorb-Repräsentation des Artikels ein entsprechendes Kennzeichen gesetzt und dieses über die relevanten Schnittstellen und Datenformate in den Vorgang der Bestellabwicklung übergeben.

Typische Anwendungsfälle für diese Kategorie sind:

- Planungruppen und sonstige komposite Artikel, die andere Artikel aus planerischen oder sonstigen Gesichtspunkten zusammenfassen.
- Instanzen, die für planerische und sonstige Hilfszwecke vorgesehen sind (Placeholder, Dummies).

Im ERP–System des Herstellers ist die Artikelnummer eines Pseudo-Artikels typischerweise nicht vorhanden, sondern wird im Rahmen der OFML–Datenanlage künstlich angelegt. In diesem Fall muss/sollte bei dem Hersteller nachgefragt werden, ob die Kategorie gesetzt werden muss.

Falls die Artikelnummer im ERP–System des Herstellers bekannt ist, dient sie evtl. auch dort nur Hilfzwecken, z.B. zur Abbildung von Typicals. Hier muss der Hersteller festlegen, ob die Kategorie für eine ordentliche Bestellabwicklung gesetzt sein muss.

An Instanzen, die der Kategorie *@PseudoArticle* angehören, bestehen folgende Anfordungen:

1. Die Instanz muss auch der Schnittstellen-Kategorie *@IF_Article* angehören.
2. Die Methode *getArticlePrice()* (Abschn. 2.4) darf keinen Preis liefern, d.h., der Rückgabewert muss vom Typ *Void* sein. (Ist diese Voraussetzung nicht erfüllt, wird die Kategorie für die gegebene Artikelinstanz ignoriert.)

OFML–Applikationen zeigen für Instanzen dieser Kategorie keinen Preis und keine Artikelnummer an, ggf. aber den Artikelkurztext[39]. Des Weiteren werden Artikel dieser Kategorie von der kaufmännischen Kalkulation ausgeschlossen.

Siehe auch nachfolgende Kategorien *@NonOrderArticle* und *@NonOfferArticle*.

`@NonOrderArticle`

Die Instanz repräsentiert einen Pseudo–Artikel, der nicht für die Bestellabwicklung relevant ist.

Gehört eine Artikelinstanz dieser Kategorie an, wird in der persistenten Warenkorb-Repräsentation des Artikels ein entsprechendes Kennzeichen gesetzt. In gedruckte Bestellungen als auch in Dokumente zum elektronischen Bestell–Datenaustausch (z.B. OEX–Dokumentart ORDERS [oex]) werden Artikel mit diesem Kennzeichen *nicht* übernommen[40].

Für Artikelinstanzen dieser Kategorie wird vorausgesetzt, dass sie (auch) der Kategorie *@PseudoArticle* angehören. (Ist diese Voraussetzung nicht erfüllt, wird die Kategorie für die gegebene Artikelinstanz ignoriert.)

Die Festlegung dieser Kategerie für einen gegebenen Artikel muss in Absprache mit dem Hersteller getroffen werden.

Ist die Kategorie für einen kompositen Artikel gesetzt, werden dessen Unterartikel im Bestell-Dokument auf die Ebene des kompositen Artikels hochgezogen.

[39] falls vorhanden

[40] Die Kategorie hat also keinen Einfluß auf Warenkorb-Sichten (Artikellisten), die nicht explizit als *Bestellung* typisiert sind.

`@NonOfferArticle`

Die Instanz repräsentiert einen Pseudo–Artikel, der nicht in Kundenangeboten ausgewiesen werden soll.

Gehört eine Artikelinstanz dieser Kategorie an, wird in der persistenten Warenkorb-Repräsentation des Artikels ein entsprechendes Kennzeichen gesetzt. In gedruckte Angebote als auch in Dokumente zum elektronischen Angebot–Datenaustausch (z.B. OEX–Dokumentart QUOTES [oex]) werden Artikel mit diesem Kennzeichen *nicht* übernommen[41].

Für Artikelinstanzen dieser Kategorie wird vorausgesetzt, dass sie (auch) der Kategorie *@PseudoArticle* angehören. (Ist diese Voraussetzung nicht erfüllt, wird die Kategorie für die gegebene Artikelinstanz ignoriert.)

Die Festlegung dieser Kategerie für einen gegebenen Artikel muss in Absprache mit dem Hersteller getroffen werden.

Ist die Kategorie für einen kompositen Artikel gesetzt, werden dessen Unterartikel im Angebotsdokument auf die Ebene des kompositen Artikels hochgezogen.

`@SALESONLY_ARTICLE`

Die Instanz repräsentiert einen Artikel ohne eine spezifische Geometrie.

Mögliche Sonderbehandlungen für Artikelinstanzen dieser Kategorie sind:

- Ausblenden beim Generieren des Artikelbildes für die übergeordnete Artikelinstanz (falls vorhanden)
- Keine dynamische Erzeugung eines Artikelbildes[42]

[41] Die Kategorie hat also keinen Einfluß auf Warenkorb-Sichten (Artikellisten), die nicht explizit als *Angebot* typisiert sind.
[42] stattdessen Verwendung eines in den Katalogdaten hinterlegten Bildes (falls vorhanden)

C OFML–Variantencode

Für den Aufbau des OFML–Variantencodes gelten die Bestimmungen für den Variantencode des vordefinierten OCD–Codierungsschemas `KeyValueList` [ocd], und zwar:

Jedes aktuell gültige/sichtbare konfigurierbare Merkmal wird in der Reihenfolge gemäß der OCD–Property–Tabelle wie folgt dargestellt:

`<Merkmalsklasse>.<Merkmal>=<Merkmalswert>`

Als Trennzeichen zwischen den Merkmalen wird das Semikolon verwendet.

Für aktuell nicht bewertete optionale und einschränkbare Merkmale wird das Wertkürzel „`VOID`“ verwendet. Bei der Codierung der Werte von bewerteten Merkmalen erfolgt keine Auffüllung mit Leerzeichen gemäß der Angabe im Längenfeld der Property–Tabelle, d.h. es werden nur die signifikanten Stellen dargestellt.

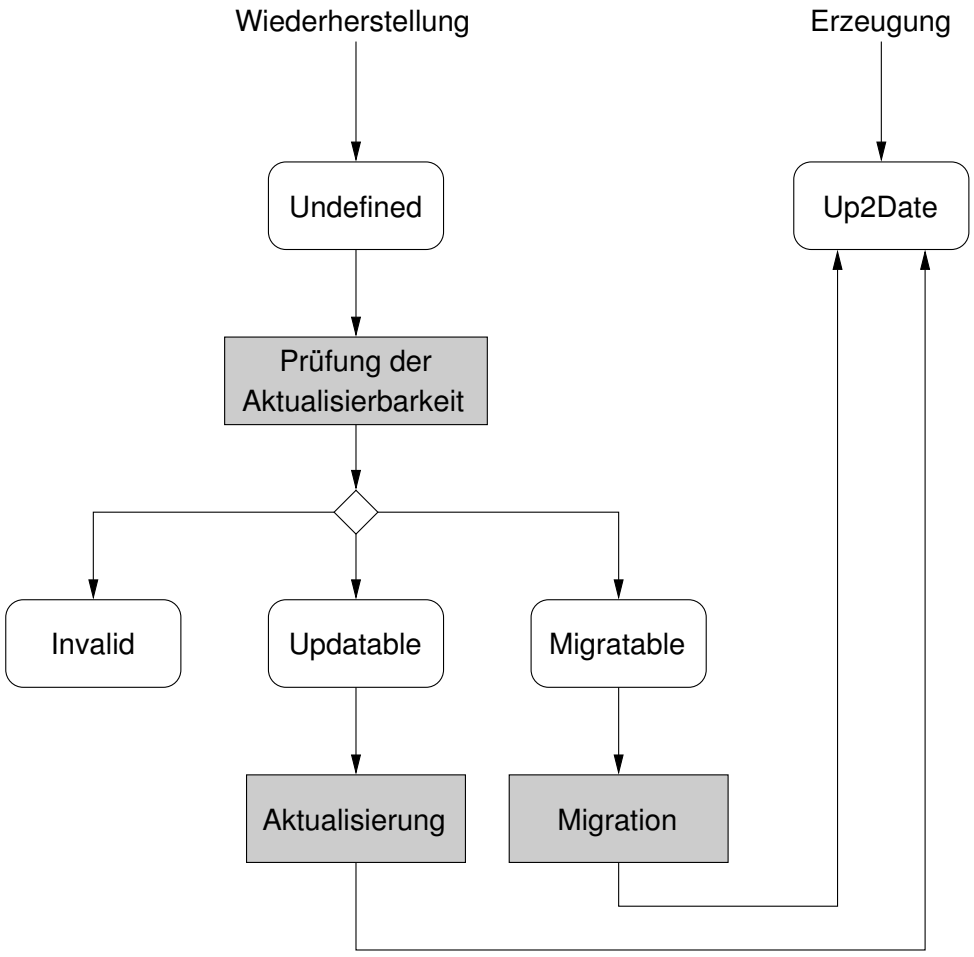
Wird der Hersteller-unabhängige OFML–Variantencode, wie im Abschn. 2.6 empfohlen, für die Aktualisierung von gespeicherten Artikelkonfigurationen verwendet, kann folgene Problematik auftreten[43]:

Wenn die OCD–Daten aus einem ERP- bzw. PPS-System exportiert werden, welches das Konzept der Merkmalsklasse nicht kennt, und wo durch die Export-Routine willkürliche Merkmalsklassen generiert werden, kann die Aktualisierung fehlschlagen, da die Merkmale aufgrund einer geänderten Merkmalsklasse eventuell nicht „erkannt“ werden.

OFML–Applikationen müssen deswegen eine alternative Überprüfung der Aktualisierbarkeit anbieten und unterstützen, bei der Merkmalsklassen im Variantencode ignoriert werden.

[43] Die Problematik tritt auch mit Hersteller-spezifischen Variantencodes auf, welche mit dem vordefinierten OCD–Codierungsschema `KeyValueList` generiert werden.

D Aktualisierungszustand



E Initialisierung von Artikelinstanzen

Beim **Einfügen** eines Artikels **aus dem Katalog** einer OFML–Applikation wird folgender Ablauf bezüglich der Initialisierung der Artikelinstanz erwartet/vorausgesetzt:

1. Definition des Programm-Kontextes.

   Die OFML–Programm–ID wird durch die Applikation aus den Katalogdaten gelesen. Aus den Registrierungsdaten für das installierte OFML–Paket mit der Programm–ID werden der Typ für das Programm–Infoobjekt (Basistyp *OiProgInfo*) und der Typ der Produktdatenbank (Basistyp *OiProductDB*) ermittelt.

   Mit diesen Angaben wird durch entsprechende Methodenaufrufe auf der globalen Planungsinstanz der Programm-Kontext hergestellt, siehe Funktionen *setProgram()*, *addInfo()* und *addProductDB()* des Basistyps *OiPlanning* in [ofml].

2. Erzeugung der Artikelinstanz.

   Der zu verwendende Typ wird anhand der aus den Katalogdaten gelesenen Grundartikelnummer mittels Aufruf der Methode *article2Class()* auf der globalen Planungsinstanz ermittelt (s. Basistyp *OiPlanning* in [ofml])[44].

   Dieser Schritt beinhaltet auch den Aufruf der Initialisierungsfunktion *initialize()* für die erzeugte Instanz.

3. Kaufmännische Initialisierung.

   Dies erfolgt durch Zuweisung von Grundartikelnummer und (möglicherweise leerem bzw. teilbestimmtem) Hersteller-spezifischem Variantencode mittels Aufruf der Methoden *setArticleSpec()* und *setXArticleSpec()* auf der Artikelinstanz (s. Abschn. 2.2).

   Grundartikelnummer und Variantencode werden aus den Katalogdaten entnommen.

4. Sonstige Initialisierung.

   Hier können mittels der OFML–Schnittstelle *Property* spezifische Merkmalsbewertungen für die Artikelinstanz vorgenommen werden, z.B. anhand von Anwenderprofilen.

   Des Weiteren erfolgt hier mittels Funktion *setCatalogInfo()* (s. Abschn. 2.8) die Zuweisung von Informationen über den Eintrag im Katalog, der beim Einfügen des Artikels verwendet wurde.

   Abschließend wird für die erzeugte Artikelinstanz auf der globalen Planungsinstanz (Basistyp *OiPlanning*) die Methode *articleInserted()* gerufen.
   Durch die in der jeweilgen OFML–Applikation verwendeten konkreten Unterklasse von *OiPlanning* kann hier ein Event, z.B. `@ArticleInserted`, an den globalen ChangeManger (Basistyp *OiChangeManager*) gemeldet werden.

Bei der **Wiederherstellung** einer Artikelinstanz anhand einer gespeicherten **Warenkorb-Repräsentation** wird folgender Ablauf bezüglich der Initialisierung der Artikelinstanz erwartet/vorausgesetzt:

1. Definition des Programm-Kontextes.

   Wie oben, die OFML–Programm–ID wird hier aber aus der Warenkorb-Repräsentation entnommen.

2. Erzeugung der Artikelinstanz.

   Wie oben, die Grundartikelnummer wird hier aber aus der Warenkorb-Repräsentation entnommen.

3. Kaufmännische Initialisierung.

   Dies erfolgt durch Zuweisung von Grundartikelnummer und Hersteller-unabhängigem OFML–Variantencode mittels Aufruf der Methode *setupConfiguration()* auf der Artikelinstanz (s. Abschn. 2.6).

[44] Dabei werden Mappingtabellen verwendet, die im OFML Part VI (OAM) standardisiert sind [oam].

Grundartikelnummer und Variantencode werden aus der Warenkorb-Repräsentation entnommen.

Hat die Artikelinstanz nach dem Methodenaufruf nicht den Update-Zustand `@Up2Date` (s. Abschn. 2.6), ist die Wiederherstellung fehlgeschlagen (und der Vorgang wird abgebrochen).

4. Sonstige Initialisierung.

   Hier erfolgt die Zuweisung des in der Warenkorb-Repräsentation gespeicherten zusätzlichen Zustandcodes mittels Aufruf der Methode *setAddStateCode()* auf der Artikelinstanz (s. Abschn. 2.3).

   Des Weiteren erfolgt hier mittels Funktion *setCatalogInfo()* (s. Abschn. 2.8) die Zuweisung von Informationen über den Eintrag im Katalog, der beim Einfügen des Artikels verwendet wurde. (Das setzt voraus, dass diese Informationen auch in der Warenkorb-Repräsentation gespeichert sind.)

5. Initialisierung von Unterartikeln.

   Im Fall eines kompositen Artikels wird für jede Unterposition im Warenkorb die dort gespeicherte *SubArticleID* gelesen und durch Aufruf der Methode *getSubArticle()* auf der kompositen Artikelinstanz die Instanz ermittelt, die den Unterartikel mit dieser ID repräsentiert (s. Abschn. 3.1).

   Für die Unterartikelinstanzen werden die Schritte 3 bis 5 wiederholt, wobei der Aufruf von *setCatalogInfo()* (im Schritt 4) für diese Instanzen entfällt.

Die Applikation teilt der globalen Planungsinstanz (Basistyp *OiPlanning*) am Ende von Schritt 1 (in beiden Szenarien) mittels Aufruf der Methode *setCreationMode(pMode(Int))* mit, welches der Szenarien vorliegt. Die entsprechenden Werte für den Parameter sind:

`0` Einfügen eines Artikels aus dem Katalog

`1` Wiederherstellung einer Artikelinstanz anhand einer gespeicherten Warenkorb-Repräsentation

Bei der Implementierung von Funktionen der Schnittstellen *Article* und *CompositeArticle* kann dieser Modus bei Bedarf berücksichtigt werden. Der Modus kann dazu mittels Aufruf der entsprechenden Methode *getCreationMode()* auf der globalen Planungsinstanz ermittelt werden[45].

Der Modus ist insbesondere für die **Erzeugung und Initialisierung von Unterartikelinstanzen** von kompositen Artikeln relevant:

`0` In diesem Szenario erfolgt die komplette Erzeugung und Initialisierung von initialen Unterartikelinstanzen während der Methoden *setArticleSpec()* bzw. *setXArticleSpec()* der kompositen Artikelinstanz, d.h., für die initialen Unterartikel werden die Schritte 1 bis 4 des Szenarios durch die komposite Artikelinstanz ausgeführt.

`1` Da in diesem Szenario die Schritte 3 und 4 für *alle* Unterartikel eines kompositen Artikels nach der Erzeugung und Initialisierung der kompositen Artikelinstanz durch die Applikation ausgelöst werden (Schritt 5), sollten hier aus Performanzgründen während *setArticleSpec()* bzw. *setXArticleSpec()*[46] keine Unterartikelinstanzen erzeugt und initialisiert werden.
Der Schritt 2 – Erzeugung der Unterartikelinstanz inkl. Positionierung – erfolgt während der Methode *setAddStateCode()* der kompositen Artikelinstanz[47]. Dazu müssen im *AddStateCode* (s. Methode *getAddStateCode()*, Abschn. 2.3) die entsprechenden Informationen[48] zu *allen* (aktuellen) Unterartikelinstanzen gespeichert werden[49].

[45] Die globale Planungsinstanz kann mittels der globalen Funktion *oiGetPlanning()* ermittelt werden.
[46] via *setupConfiguration()*
[47] Ggf. doch während *setupConfiguration()* erzeugte Unterartikelinstanzen werden hier entsprechend wieder entfernt!
[48] mindestens also: SubArticleID, Programm-ID, Grundartikelnummer, Position, Rotation
[49] also auch zu Unterartikelinstanzen, die interaktiv durch den Anwender eingefügt bzw. geändert wurden

F Änderungshistorie

F.1 Version 1.3 vs. Version 1.2 (2024-10-21)

- Präzisierungen und kleinere stilistische Verbesserungen in der Einleitung (Abschn. 1).
- Ergänzungen im Anhang E, insbesondere zur Erzeugung und Initialisierung von Unterartikelinstanzen.

F.2 Version 1.2 vs. Version 1.1 (2024-01-11)

- Funktion *getArticleObj()* vom Abschn. 2.4 in den Abschn. 2.8 verschoben.
- Beschreibung der Möglichkeit der Nutzung von Produktdaten-Caches für gleiche Artikelkonfigurationen (Abschn. 2.4).
- Neue Funktion *getArticleClassifications()* im Abschn. 2.4.
- Neuer Anhang B enthält Spezifikationen von Standard–Kategorien.

F.3 Version 1.1 vs. Version 1.0 (2022-05-02)

- Neue Funktion *getPDLanguage()* im Abschn. 2.8.