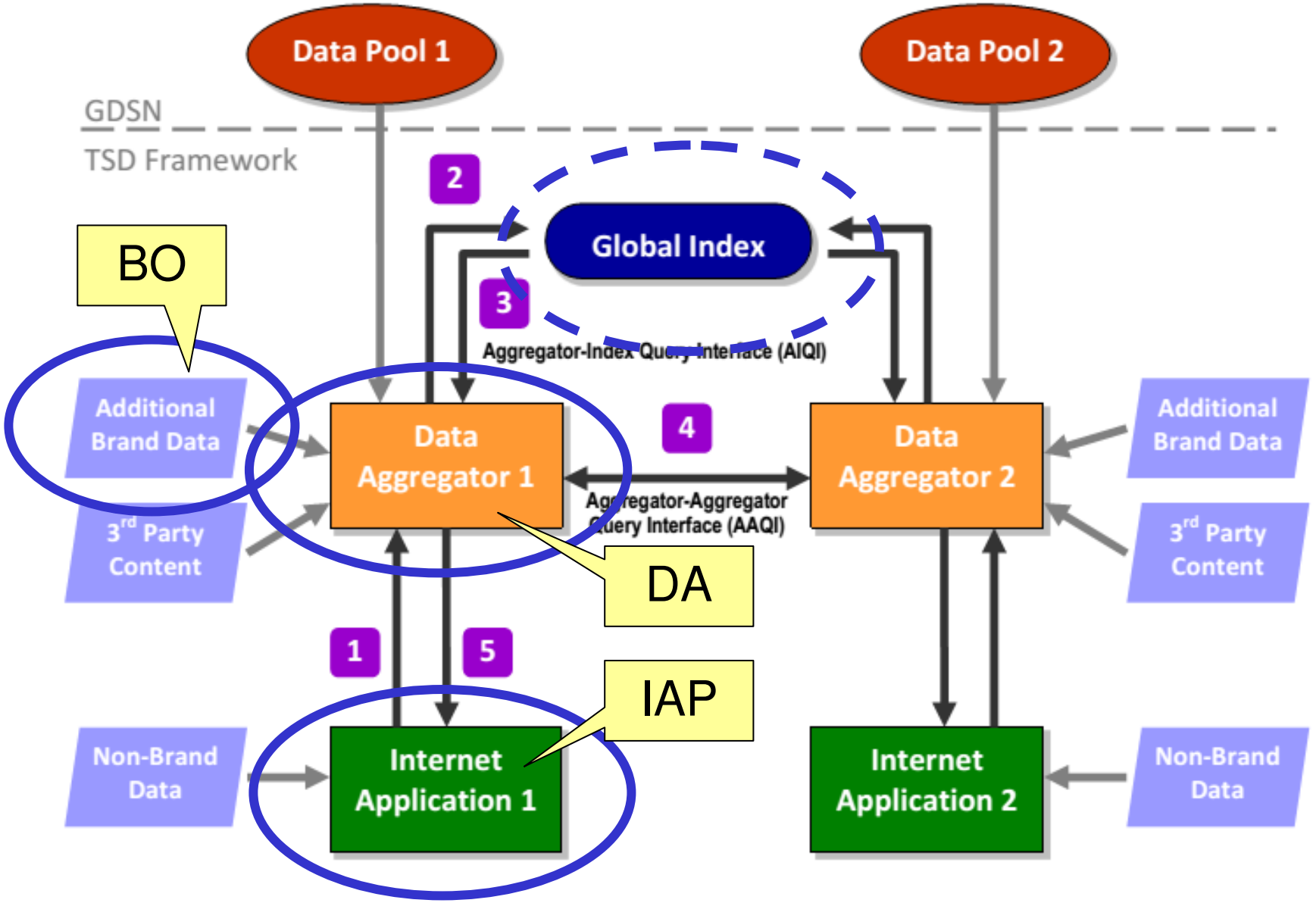
8530 - Anwendungsintegration
Projektvorgaben und -annahmen

Projektvorgaben

Annahmen

Szenario



Allgemeines

- **Team-Bildung:**
  - Gesucht werden IAP-, DA-, und BO-Teams
    - Team-Größen: IAP: 2-3, DA: 3, BO: 2-3
    - Rolle „Globaler Index“ wird vom Dozenten gestellt
  - Es muss mindestens 2 DA-Teams geben!
- **Aufgaben IAP („Internet Application Provider“)**
  - Anwendungskern mit REST-Interface zu einem DA, Caching
  - Web-Interface für Nutzer
  - Optional: App+App-Interface für Nutzer (nur Dreier-Teams)
- **Aufgaben DA („Data Aggregator“)**
  - Impl. aller Schnittstellen (DA/DA, DA/BO, DA/IAP, DA/GI), Caching
- **Aufgaben BO („Brand Owner“)**
  - Anwendungskern zur Verwaltung von Artikelstammdaten
  - REST-Interface zu einem DA, incl. Maintenance
  - Optional: Web-Oberfläche / App zur Erfassung von Artikelstammdaten (Dreier-Teams)

Allgemeines

- Grundlage:
  - Dokument GS1_Source_TSD_Standard-i1p1-Aug2013.pdf
  - Abhängige Dokumente: Product_Data_Modules, Schemata (ZIP)
- Abweichungen:
  - Inhalt
    - Es wird stets derselbe Dokumententyp ausgetauscht!
    - Beschränkung auf zwei Module („basic“ und … (noch zu besprechen))
  - REST:
    - Abweichungen im URI-Aufbau
    - dataVersion entfällt!
    - Exception-Behandlung erfolgt nur über *http status code* (und optionale Zusatzinformationen im „Body“)
    - Authentifizierungsangaben werden von URLs getrennt
    - Einheitlichkeit der Interfaces IAP / DA und DA / DA
    - Interface BO / DA kompatibel!
  - Generelle Caching-Fähigkeit ist wichtig

Abweichungen

- Diskussion:
  - „productData“ oder „TSD_QueryByGTINResponseType“ (Wrapper)
  - REST-Idee beachten, ggf.: Folgen für Fehlerbehandlung?

- Diskussion:
  - „dataVersion“: API-Lücke? Konsequenzen?
  - Hier weglassen?

- Diskussion:
  - „mac“ und „clientGln“: REST-kompatibel? Wirkung auf Caching?

  - Tipp für Ruby-Anwender:

```
require 'digest/hmac'

mac = Digest::HMAC.hexdigest("data", "hash key", Digest::SHA256)
```

Caching

- Hintergrund:
  - Frühere Versionen der GS1-Spezifikation hatten Caching ausdrücklich *untersagt*
  - Man wollte offenbar verhindert, dass veraltete Daten in Umlauf sind
  - Nun gibt es offenbar keine solchen Restriktionen mehr, Caching wird nicht einmal mehr erwähnt.
    - Einzige verbliebene Spur: productData/timeToLive
    - Neue Rezepturen (Bsp. für eine Änderung in productData) breiten sich aber nur innerhalb von Tagen entlang der Supply Chain aus
    - Ein Tag als Default-Lebensdauer für Cache-Daten ist daher kein Risiko
- Ihre Aufgabe
  - Caching ist sehr wichtig angesichts der großen Datenmengen, die im Endausbau erfragt werden können.
  - IAP-Teams sollen productData-Dokumente in einem lokalen Cache verwalten mit einer Verfallszeit von einem Tag, soweit nichts anderes vereinbart wird. Tipp: OS-Produkte nehmen, z.B. „squid“
  - Für DA-Teams gilt das gleiche. Ihre Cache-Einträge sollen aber ggf. mit dem timeToLive-Wert des BO initialisiert werden.

XML-Schema

- Schema-Abweichungen
  - Root-Element: „productData“
  - elementFormDefault="qualified"
  - Bitte beachten: Element „timeToLive“
    - Verwendbar für Initialisierung des Cachings bei der Datenbereitstellung

- **DA – DA**
  - XML-Schema wie bereitgestellt, insb. in product_data.xsd

- **IAP – DA**
  - Im Standard-Dokument nicht berücksichtigt
  - Hier: Wie Fall DA – DA behandeln!

REST-Schnittstellen

- **IAP** / **DA** und **DA** / **DA**
  - Anfrage:

    GET /*base_url*/**service**/**v1**/**product_data**/**gtin**/*gtin*?targetMarket=*XYZ*

    Ein Dokument, oder Fehler
  - Bemerkungen
    - Diskussion: Auf Parameter „dataVersion“ verzichten wegen konzeptioneller Probleme?: Welche Versionen gibt es? Wie erfährt man die? API-Lücke!?
    - Parameter *mac*: Erst berücksichtigen, wenn der Rest funktioniert
    - Parameter *clientGln*: Diskussion!

      Caching-Bremse?! Nur für mac-Berechnung erforderlich?!
    - Laut Spezifikation muss targetMarket gemäß „ISO-3166-1 numeric“ verschlüsselt werden. Beispiele: Deutschland = 276, USA = 840
    - Tabelle dazu → Wikipedia

REST-Schnittstellen

- Nur **Brand Owner / DA**
  - Neuanlage:

    POST /*base_url*/**service/v1/product_data/gtin**

    OK oder Fehler. Legt neues Dokument an

    Location Header:
    /*base_url*/**service/v1/product_data/gtin**/*gtin*?targetMarket=*XYZ*

    gtin und *XYZ* müssen dabei den Inhalten aus dem Dokument entsprechen!
  - Aktualisierung:

    PUT /*base_url*/**service/v1/product_data/gtin**/*gtin*?targetMarket=*XYZ*

    OK oder Fehler. Aktualisiert
  - Löschung:

    DELETE /*base_url*/**service/v1/product_data/gtin**/*gtin*?targetMarket=*XYZ*

    OK oder Fehler
  - Parameter *mac, clientGln*:
    - Nur zu implementieren, wenn auch bei GET (DA-DA) vorgesehen

STOP

Der Rest ist noch im Laufe des Dezember zu beschließen

Alle folgenden Angaben können sich noch erheblich ändern!

REST-Schnittstellen

- Security-Aspekte (IAP/DA, DA/DA, Brand Owner/DA)
  - Details werden zwischen den DA-Teams verhandelt und festgelegt
  - Die IAP/Brand Owner-Teams erfahren das Ergebnis zeitnah und richten sich danach
  - Wichtig: Einheitliches Vorgehen
  - Materialsammlung:
    - http://www.heise.de/developer/artikel/Identity-Management-Authentifizierungsdienste-mit-OpenID-227202.html
    - http://www.heise.de/developer/artikel/Step2-Protokoll-OpenID-und-OAuth-Hand-in-Hand-1359904.html
    - http://www.openid.net/connect/
    - http://tools.ietf.org/html/rfc6749
    - http://hueniverse.com/2010/09/oauth-bearer-tokens-are-a-terrible-idea/
    - http://hueniverse.com/2012/07/oauth-2-0-and-the-road-to-hell/
- Security-Aspekte (DA/Global Index)
  - Stark vereinfacht, siehe DA/Index-Schnittstelle

REST-Schnittstellen

- DA / Index: AIQI
  - Grundsätzliches
    - Die DA/Index-Schnittstelle ist RESTful. Ein Eintrag als abstrakte Ressource besteht aus folgenden Feldern

`id`	Numerisches Feld (Integer, >= 0)
`gtin`	Numerisches Feld, stets 14-stellig
`target_market`	Numerisches Feld, stets 3-stellig
`data_aggregator_service`	String-Feld, nur URLs erlaubt

    - Für den Austausch wird nur die JSON-Darstellung unterstützt.
    - Feld „id“ wird vom Index-Server vergeben. Neue Einträge dürfen „id“ nicht enthalten.
    - Update- und Delete-Schritte erfordern die Kenntnis der zugehörigen „id“. Index-Nutzer merken sich daher die ids zu „ihren“ Einträgen, oder sie ermitteln sie zunächst per Lese-Zugriff
    - Abfragen erfolgen i.d.R. per GTIN und *target market*. Sie ergeben daher Arrays von Treffern. Diese können auch leer sein!

REST-Schnittstellen

- DA / Index: AIQI
  - Normaler Lese-Zugriff
    - **GET** */base_url/***v1**/**index_entries**?gtin=*gtin*&targetMarket=*XYZ*
    - Beispiel:
      GET /tsd/v1/index_entries?
      gtin=*1234567890123***&targetMarket=***276*

    - Antwort: JSON-Datei. Beispiel:

```
Content-Type: application/json

[{ "gtin": 1234567890123, "targetMarket":276,
  "id": 1, "dataAggregatorService":
  "http://www.some_server.xy/base/" }]
```

  - Bemerkungen
    - Inhalt ist immer gleich aufgebaut, auch wenn Redundanz besteht zwischen Anfrage und Antwort
    - Parameter: „gtin“ ist Pflicht, „targetMarket“ ist in GET optional

REST-Schnittstellen

- DA / Index: AIQI
  - Normaler Lese-Zugriff
    - **GET** */base_url/***service/v1/index_entries**?
      gtin=*gtin*&targetMarket=*XYZ*
    - Beispiel:
      GET /tsd/service/v1/index_entries?
      gtin=*123456890123***&targetMarket=***276***
    - Antwort: JSON-Datei. Beispiel:

```
Content-Type: application/json

[{ "gtin": 1234567890123, "targetMarket":276,
  "id": 1, "dataAggregatorService":
  "http://www.some_server.xy/base/" }]
```

  - Bemerkungen
    - Inhalt ist immer gleich aufgebaut, auch wenn Redundanz besteht zwischen Anfrage und Antwort
    - Parameter: „gtin“ ist Pflicht, „targetMarket“ ist in GET optional

REST-Schnittstellen

- DA / Index: AIQI
  - Fehler, allgemein
    - Zusätzliche Parameter

      Keine Fehlermeldung, sie werden einfach ignoriert
    - Falsch aufgebaute URLs

      400 Bad Request
    - Nicht unterstützte Darstellung angefragt (z.B. XML)

      415 Unsupported Media Type
    - Autorisierungsfehler

      Siehe unten
  - Fehler, spezifisch
    - GTIN fehlt

      403 Forbidden

REST-Schnittstellen

- DA / Index: AIQI
  - Direkter Lese-Zugriff
    - **GET** */base_url*/**service**/**v1**/**index_entries**/*id*
    - Beispiel:
      GET /tsd/**service/v1/index_entries/***1465493234*

    - Antwort: JSON-Datei. Beispiel:

```
Content-Type: application/json

{ "gtin": 1234567890123, "target_market":276,
  "id": 1465493234, "data_aggregator_service":
  "http://www.some_server.xy/base/" }
```

  - Bemerkungen
    - Antwort: Nur Hash, kein Array von Hashes
    - 404 Not Found wenn id unbekannt

REST-Schnittstellen

- DA / Index: AIQI
  - Security: Access token/bearer token-Prinzip
    - Jedes DA-Team meldet dem Index-Betreiber „seinen“ Basis-URL
    - Jedes DA-Team erhält zum Basis-URL ein geheimes „bearer token“
    - Dieses „bearer token“ ist assoziiert mit dem hinterlegten URL des DA
    - DA-Zugriffe auf den Index übermitteln das Token im HTTP Header:

      ```
      Authorization: Bearer asd8979addfg67
      ```

    - Einfache Kontrolle bei GET: „secret“ muss dem Server bekannt sein.
    - Falls fehlerhafter Code, oder falls fehlender Header:

      ```
      401 Unauthorized
      ```

    - Vereinfachung: Kein WWW-Authenticate-Header, kein „Realm“, „langlebige“ *tokens*
  - Fehler-Codes
    - 200 OK
    - 401 Unauthorized Auth-Header fehlt oder kein gültiges Token

REST-Schnittstellen

- DA / Index: AIMI
  - Neuen Eintrag anlegen
    - **POST** /*base_url*/**service**/**v1**/**index_entries**
    - Body ist die o.g. JSON-Struktur (ohne id)
    - Das Paar (gtin, targetMarket) darf nicht bereits existieren
    - GTIN muss gültig sein (Prüfziffer!), targetMarket im Bereich 100..999
    - Allgemein, bei Validierungsfehlern

      422 Unprocessable Entity, Fehlermeldung im Body (application/json)
    - Security:

      Zum Bearer token hinterlegter URL muss gleich dem in JSON-Feld „data_aggregator_service“ sein!

      Sonst: 401 Unauthorized, Fehlermeldung im Body (text/plain)
    - Im Erfolgsfall:

      201 Created

      Location-Header enthält URL zur neuen Ressource, insb. die Id

REST-Schnittstellen

- DA / Index: AIMI
  - Eintrag löschen
    - **DELETE** /*base_url*/**service/v1/index_entries/*id***
    - Security:

      Zum Bearer token hinterlegter URL muss gleich dem in „data_aggregator_service“ enthaltenen URL sein!

      Grund: Einträge dürfen nur vom erzeugenden DA gelöscht werden

      Falls nicht: 401 Unauthorized, Fehlertext im Body (text/plain)
    - Sonstige Fehler:

      Falls zu id kein Eintrag vorhanden:

      404 Not Found

REST-Schnittstellen

- DA / Index: AIMI
  - Eintrag ändern
    - Nicht wirklich sinnvoll, wird daher nicht unterstützt
    - Ersatzweise empfohlen: Alten Eintrag löschen, neuen anlegen
    - Konsequenz: „PUT“ nicht erlaubt → 405 Method not allowed
  - Falls Wechsel des „dataAggregatorService“-URLs erforderlich:
    - DA benötigt zwei Identitäten (eine pro URL)
    - DA liest & löscht über „alten“ URL-Wert / altes Token
    - DA erzeugt neuen Eintrag über „neuen“ URL-Wert / Token