Data Schemas

Faculty of Computer Science
Workflow Systems and Technologies

Interoperability

Schema for Semi-Structured Data

- A schema defines the formal structure of data.
- Enables to specify data types and constraints for the content.
- Typical use cases are documentation and validation of data.
- XML is supported by
  - Document Type Definition (DTD)
  - XML Schema Definition (XSD) (W3C recommendation)
  - REgular LAnguage for XML Next Generation (RelaxNG)
- while JSON is supported by JSON Schema.

Basics of selected XML and JSON Schema Languages

DTD

```
<!ELEMENT payment (
  transaction-id, status, total,
  datetime, order-items)>
<!ELEMENT transaction-id (#PCDATA)>
<!ELEMENT status (#PCDATA)>
<!ELEMENT total (#PCDATA)>
<!ELEMENT datetime (#PCDATA)>
<!ELEMENT order-items (item+)>
<!ELEMENT item (#PCDATA)>
<!ATTLIST total currency CDATA #FIXED 'EUR'>
<!ATTLIST item qty CDATA #REQUIRED>
<!ATTLIST item vat CDATA #REQUIRED>
<!ATTLIST item price CDATA #REQUIRED>
<!ATTLIST item name CDATA #REQUIRED>
<!ENTITY ok "OK">
```

```
<payment>
  <transaction-id>#d9w72d83</transaction-id>
  <status>OK</status>
  <total currency="EUR">26.56</total>
  <datetime>2024-04-10T09:45:00</datetime>
  <order-items>
    <item qty="1"
          vat="10%"
          price="20.00"
          name="Cheese Cake 1kg"/>
    <item qty="1"
          vat="20%"
          price="3.80"
          name="Cement 20kg"/>
  </order-items>
</payment>
```

DTD

```
...
<!ELEMENT payment (
  transaction-id, status, total,
  datetime, order-items)>
...
```

!ELEMENT defines the name of an element's tag and defines its children in the expected order as a comma-separated list surrounded by parentheses.

```
<payment>
  <transaction-id>#d9w72d83</transaction-id>
  <status>OK</status>
  <total currency="EUR">26.56</total>
  <datetime>2024-04-10T09:45:00</datetime>
  <order-items>
    <item qty="1"
          vat="10%"
          price="20.00"
          name="Cheese Cake 1kg"/>
    <item qty="1"
          vat="20%"
          price="3.80"
          name="Cement 20kg"/>
  </order-items>
</payment>
```

DTD

```
…
<!ELEMENT transaction-id (#PCDATA)>
…
```

Defines an element which only contains text.

```
<payment>
  <transaction-id>#d9w72d83</transaction-id>
  <status>OK</status>
  <total currency="EUR">26.56</total>
  <datetime>2024-04-10T09:45:00</datetime>
  <order-items>
    <item qty="1"
          vat="10%"
          price="20.00"
          name="Cheese Cake 1kg"/>
    <item qty="1"
          vat="20%"
          price="3.80"
          name="Cement 20kg"/>
  </order-items>
</payment>
```

DTD

```
…
<!ATTLIST total currency CDATA #FIXED 'EUR'>
<!ATTLIST item qty CDATA #REQUIRED>
…
```

!ATTLIST specifies the attributes of an element. *CDATA* stands for 'character data'. *#FIXED* defines a constraint that ensures the attribute's value is set to the constant specified (in this case 'EUR'). *#REQUIRED* ensures the attribute is present.

```
<payment>
  <transaction-id>#d9w72d83</transaction-id>
  <status>OK</status>
  <total currency="EUR">26.56</total>
  <datetime>2024-04-10T09:45:00</datetime>
  <order-items>
    <item qty="1"
          vat="10%"
          price="20.00"
          name="Cheese Cake 1kg"/>
    <item qty="1"
          vat="20%"
          price="3.80"
          name="Cement 20kg"/>
  </order-items>
</payment>
```

DTD

```
…
<!ELEMENT order-items (item+)>
…
```

Defines the cardinality of the child element 'item' as + (one or more). Cardinalities can be defined per child element and via ()+ for the parent element. Other options include ? (none or one), * (zero or more) or - (once).

```
<payment>
  <transaction-id>#d9w72d83</transaction-id>
  <status>OK</status>
  <total currency="EUR">26.56</total>
  <datetime>2024-04-10T09:45:00</datetime>
  <order-items>
    <item qty="1"
          vat="10%"
          price="20.00"
          name="Cheese Cake 1kg"/>
    <item qty="1"
          vat="20%"
          price="3.80"
          name="Cement 20kg"/>
  </order-items>
</payment>
```

XSD

```
<schema xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="payment">
    <complexType>
      <sequence>
        <element name="transaction-id" type="string" />
        <element name="status">
          <simpleType>
            <restriction base="string">
              <enumeration value="OK"/>
              <enumeration value="NOK"/>
            </restriction>
          </simpleType>
        </element>
        <element name="total">
          <complexType>
            <simpleContent>
              <extension base="decimal">
                <attribute name="currency" type="string" use="required" />
              </extension>
            </simpleContent>
          </complexType>
        </element>
        <element name="datetime" type="dateTime" />
        <element name="order-items">
          <complexType>
            <sequence>
              <element name="item" minOccurs="1" maxOccurs="unbounded">
                <complexType>
                  <attribute name="qty" type="integer" use="required" />
                  <attribute name="vat" use="required">
                    <simpleType>
                      <restriction base="string">
                        <pattern value="[0-9]+%"/>
                      </restriction>
                    </simpleType>
                  </attribute>
                  <attribute name="price" type="decimal" use="required" />
                  <attribute name="name" type="string" use="required" />
                </complexType>
              </element>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>
```

```
<payment>
  <transaction-id>#d9w72d83</transaction-id>
  <status>OK</status>
  <total currency="EUR">26.56</total>
  <datetime>2024-04-10T09:45:00</datetime>
  <order-items>
    <item qty="1"
          vat="10%"
          price="20.00"
          name="Cheese Cake 1kg"/>
    <item qty="1"
          vat="20%"
          price="3.80"
          name="Cement 20kg"/>
  </order-items>
</payment>
```

XSD

```
<schema xmlns="http://www.w3.org/2001/XMLSchema">
...
</schema>
```

The schema is defined in the root tag 'schema'. Note the default namespace (xmlns) which provides access to XSD's building blocks such as data types for defining schemas.

```
<payment>
  <transaction-id>#d9w72d83</transaction-id>
  <status>OK</status>
  <total currency="EUR">26.56</total>
  <datetime>2024-04-10T09:45:00</datetime>
  <order-items>
    <item qty="1"
          vat="10%"
          price="20.00"
          name="Cheese Cake 1kg"/>
    <item qty="1"
          vat="20%"
          price="3.80"
          name="Cement 20kg"/>
  </order-items>
</payment>
```

XSD

```
…
<element name="transaction-id"
         type="string"/>
…
<element name="datetime" type="dateTime"/>
…
```

This simple *element* definition represent a XML element named *transaction-id* which only permits text-based content (not to be confused with the data type string) and disallows nesting further elements.

The *type* attribute enables to define how the text content should be formatted. See the XML Schema specification for the list of predefined types. Custom types are also permitted.

```
<payment>
  <transaction-id>#d9w72d83</transaction-id>
  <status>OK</status>
  <total currency="EUR">26.56</total>
  <datetime>2024-04-10T09:45:00</datetime>
  <order-items>
    <item qty="1"
          vat="10%"
          price="20.00"
          name="Cheese Cake 1kg"/>
    <item qty="1"
          vat="20%"
          price="3.80"
          name="Cement 20kg"/>
  </order-items>
</payment>
```

XSD

```
…
<element name="payment">
  <complexType>
    <sequence>
      <element name="transaction-id"
               type="string"/>
      <element name="status">
      …
    …
    </sequence>
  </complexType>
</element>
…
```

Elements that are required to contain children or carry attributes must be defined as *complexType* (the alternative is *simpleType*). *sequence* defines the children of the element and the order in which the child elements should appear.

```
<payment>
  <transaction-id>#d9w72d83</transaction-id>
  <status>OK</status>
  <total currency="EUR">26.56</total>
  <datetime>2024-04-10T09:45:00</datetime>
  <order-items>
    <item qty="1"
          vat="10%"
          price="20.00"
          name="Cheese Cake 1kg"/>
    <item qty="1"
          vat="20%"
          price="3.80"
          name="Cement 20kg"/>
  </order-items>
</payment>
```

XSD

```
…
<element name="total">
  <complexType>
    <simpleContent>
      <extension base="decimal">
        <attribute name="currency"
                   type="string"
                   use="required"/>
      </extension>
    </simpleContent>
  </complexType>
</element>
…
```

The *attribute* element here defines an attribute named *currency* that should appear on the element *total.* With the attribute *use* we can define if the attribute in question must be provided. *simpleContent* enables the element *total* to contain text and to define *extensions* and *restrictions*. The *extension*'s *base* attribute determines the type of the element *total*.

```
<payment>
  <transaction-id>#d9w72d83</transaction-id>
  <status>OK</status>
  <total currency="EUR">26.56</total>
  <datetime>2024-04-10T09:45:00</datetime>
  <order-items>
    <item qty="1"
          vat="10%"
          price="20.00"
          name="Cheese Cake 1kg"/>
    <item qty="1"
          vat="20%"
          price="3.80"
          name="Cement 20kg"/>
  </order-items>
</payment>
```

XSD

```
…
<element name="status">
  <simpleType>
    <restriction base="string">
      <enumeration value="OK"/>
      <enumeration value="NOK"/>
    </restriction>
  </simpleType>
</element>
…
```

A *restriction* enables to define constraints on the content of the *status* element. Here we restrict the pool of acceptable values to *(OK, NOK)* using *enumerations.*

```
<payment>
  <transaction-id>#d9w72d83</transaction-id>
  <status>OK</status>
  <total currency="EUR">26.56</total>
  <datetime>2024-04-10T09:45:00</datetime>
  <order-items>
    <item qty="1"
          vat="10%"
          price="20.00"
          name="Cheese Cake 1kg"/>
    <item qty="1"
          vat="20%"
          price="3.80"
          name="Cement 20kg"/>
  </order-items>
</payment>
```

XSD

```
…
<element name="status">
  <simpleType>
    <restriction base="string">
      <enumeration value="OK"/>
      <enumeration value="NOK"/>
    </restriction>
  </simpleType>
</element>
…
```

A *restriction* enables to define constraints on the content of the *status* element. Here we restrict the pool of acceptable values to *(OK, NOK)* using *enumerations.*

```
<payment>
  <transaction-id>#d9w72d83</transaction-id>
  <status>OK</status>
  <total currency="EUR">26.56</total>
  <datetime>2024-04-10T09:45:00</datetime>
  <order-items>
    <item qty="1"
          vat="10%"
          price="20.00"
          name="Cheese Cake 1kg"/>
    <item qty="1"
          vat="20%"
          price="3.80"
          name="Cement 20kg"/>
  </order-items>
</payment>
```

XSD

```
…
<attribute name="vat"
           use="required">
   <simpleType>
     <restriction base="string">
       <pattern
          value="[0-9]+%"/>
     </restriction>
   </simpleType>
 </attribute>
 …
```

```
<payment>
  <transaction-id>#d9w72d83</transaction-id>
  <status>OK</status>
  <total currency="EUR">26.56</total>
  <datetime>2024-04-10T09:45:00</datetime>
  <order-items>
    <item qty="1"
          vat="10%"
          price="20.00"
          name="Cheese Cake 1kg"/>
    <item qty="1"
          vat="20%"
          price="3.80"
          name="Cement 20kg"/>
  </order-items>
</payment>
```

Alternatively we can also specify the desired pattern for strings using regular expressions as a *restriction*. In the example above we expect the value of the attribute *vat* to be formatted as an integer followed by the symbol %.

XSD

```
…
<element name="order-items">
   <complexType>
     <sequence>
       <element name="item"
                minOccurs="1"
                maxOccurs="unbounded">
       …
       </element>
     </sequence>
   </complexType>
</element>
…
```

minOccurs and *maxOccurs* quantify the expected number of elements. minOccurs by default is 1. In the example above the element *order-items* can hold 1 or more *item* elements.

```
<payment>
  <transaction-id>#d9w72d83</transaction-id>
  <status>OK</status>
  <total currency="EUR">26.56</total>
  <datetime>2024-04-10T09:45:00</datetime>
  <order-items>
    <item qty="1"
          vat="10%"
          price="20.00"
          name="Cheese Cake 1kg"/>
    <item qty="1"
          vat="20%"
          price="3.80"
          name="Cement 20kg"/>
  </order-items>
</payment>
```

RelaxNG

```xml
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

  <start>
    <ref name="payment"/>
  </start>

  <define name="payment">
    <element name="payment">
      <element name="transaction-id"><text/></element>
      <element name="status">
        <choice>
          <value>OK</value>
          <value>NOK</value>
        </choice>
      </element>
      <element name="total">
        <data type="decimal"/>
        <attribute name="currency">
          <choice>
            <value>EUR</value>
          </choice>
        </attribute>
      </element>
      <element name="datetime">
        <data type="dateTime"/>
      </element>
      <ref name="order-items"/>
    </element>
  </define>

  <define name="order-items">
    <element name="order-items">
      <oneOrMore>
        <element name="item">
          <attribute name="qty">
            <data type="integer"/>
          </attribute>
          <attribute name="vat">
            <data type="string">
              <param name="pattern">[0-9]+%</param>
            </data>
          </attribute>
          <attribute name="price">
            <data type="decimal"/>
          </attribute>
          <attribute name="name"/>
        </element>
      </oneOrMore>
    </element>
  </define>
</grammar>
```

```xml
<payment>
  <transaction-id>#d9w72d83</transaction-id>
  <status>OK</status>
  <total currency="EUR">26.56</total>
  <datetime>2024-04-10T09:45:00</datetime>
  <order-items>
    <item qty="1"
          vat="10%"
          price="20.00"
          name="Cheese Cake 1kg"/>
    <item qty="1"
          vat="20%"
          price="3.80"
          name="Cement 20kg"/>
  </order-items>
</payment>
```

RelaxNG

```
<grammar
  xmlns="http://relaxng.org/ns/structure/1.0"
  datatypeLibrary=
    "http://www.w3.org/2001/XMLSchema-datatypes">

</grammar>
```

grammar contains the schema description. The default namespace points to the RelaxNG schema specification.

RelaxNG does not provide data type support as part of its specification. It, however, does support external specifications which can be referenced via the attribute *datatypeLibrary*. All nested children of the parent with a *datatypeLibrary* inherit the definition.

```
<payment>
  <transaction-id>#d9w72d83</transaction-id>
  <status>OK</status>
  <total currency="EUR">26.56</total>
  <datetime>2024-04-10T09:45:00</datetime>
  <order-items>
    <item qty="1"
          vat="10%"
          price="20.00"
          name="Cheese Cake 1kg"/>
    <item qty="1"
          vat="20%"
          price="3.80"
          name="Cement 20kg"/>
  </order-items>
</payment>
```

RelaxNG

```
<start>
  <ref name="payment"/>
</start>

<define name="payment">
…
</define>

<define name="order-items">
…
</define>
```

start defines which schema definition should be applied starting at the root element of a document. The schema definitions can be modularized using *define*. The *ref* element enables to import schema definitions contained in *define* elements.

```
<payment>
  <transaction-id>#d9w72d83</transaction-id>
  <status>OK</status>
  <total currency="EUR">26.56</total>
  <datetime>2024-04-10T09:45:00</datetime>
  <order-items>
    <item qty="1"
          vat="10%"
          price="20.00"
          name="Cheese Cake 1kg"/>
    <item qty="1"
          vat="20%"
          price="3.80"
          name="Cement 20kg"/>
  </order-items>
</payment>
```

RelaxNG

```
…
<element name="transaction-id">
  <text/>
</element>
…
```

This defines a simple element which can contain text-only content. In order to specify an empty element one can use *<empty/>* in place of *<text/>*.

```
<payment>
  <transaction-id>#d9w72d83</transaction-id>
  <status>OK</status>
  <total currency="EUR">26.56</total>
  <datetime>2024-04-10T09:45:00</datetime>
  <order-items>
    <item qty="1"
          vat="10%"
          price="20.00"
          name="Cheese Cake 1kg"/>
    <item qty="1"
          vat="20%"
          price="3.80"
          name="Cement 20kg"/>
  </order-items>
</payment>
```

RelaxNG

```
…
<element name="total">
  <data type="decimal"/>
  <attribute name="currency">
    <choice>
      <value>EUR</value>
    </choice>
  </attribute>
</element>
<element name="datetime">
  <data type="dateTime"/>
</element>
…
```

To specify the data type of an element use the element *data*, which in this case references the type specified by the datatypeLibrary.

Attributes are defined with *attribute*.

```
<payment>
  <transaction-id>#d9w72d83</transaction-id>
  <status>OK</status>
  <total currency="EUR">26.56</total>
  <datetime>2024-04-10T09:45:00</datetime>
  <order-items>
    <item qty="1"
          vat="10%"
          price="20.00"
          name="Cheese Cake 1kg"/>
    <item qty="1"
          vat="20%"
          price="3.80"
          name="Cement 20kg"/>
  </order-items>
</payment>
```

RelaxNG

```
…
<element name="status">
  <choice>
    <value>OK</value>
    <value>NOK</value>
  </choice>
</element>
…
```

Enumerations can be expressed with *choice* for both attributes and elements.

choice can also be used to provide alternative match options for elements and attributes in addition to simple values.

Also see *group*, which allows to treat elements or attributes as a unit just like in the regular expression (a|(ab)|c).

```
<payment>
  <transaction-id>#d9w72d83</transaction-id>
  <status>OK</status>
  <total currency="EUR">26.56</total>
  <datetime>2024-04-10T09:45:00</datetime>
  <order-items>
    <item qty="1"
          vat="10%"
          price="20.00"
          name="Cheese Cake 1kg"/>
    <item qty="1"
          vat="20%"
          price="3.80"
          name="Cement 20kg"/>
  </order-items>
</payment>
```

RelaxNG

```
…
<attribute name="vat">
   <data type="string">
     <param name="pattern">[0-9]+%</param>
   </data>
 </attribute>
 …
```

The data type *string* offers additional restriction parameters. In the example above the parameter *pattern* enables us to specify the expected format for the value of the attribute *vat*.

```
<payment>
  <transaction-id>#d9w72d83</transaction-id>
  <status>OK</status>
  <total currency="EUR">26.56</total>
  <datetime>2024-04-10T09:45:00</datetime>
  <order-items>
    <item qty="1"
          vat="10%"
          price="20.00"
          name="Cheese Cake 1kg"/>
    <item qty="1"
          vat="20%"
          price="3.80"
          name="Cement 20kg"/>
  </order-items>
</payment>
```

RelaxNG

```
<element name="payment">
  <element name="transaction-id">
    <text/>
  </element>
  <element name="status">
    …
  </element>
  ..
  <ref name="order-items"/>
</element>
```

To define hierarchies of nested structures one can directly nest elements under other elements. Note the use of *ref* which is replaced by the definition it points to.

```
<payment>
  <transaction-id>#d9w72d83</transaction-id>
  <status>OK</status>
  <total currency="EUR">26.56</total>
  <datetime>2024-04-10T09:45:00</datetime>
  <order-items>
    <item qty="1"
          vat="10%"
          price="20.00"
          name="Cheese Cake 1kg"/>
    <item qty="1"
          vat="20%"
          price="3.80"
          name="Cement 20kg"/>
  </order-items>
</payment>
```

RelaxNG

```
…
<oneOrMore>
  …
</oneOrMore>
…
```

oneOrMore defines that an element or attribute must appear at least once. Alternative: zeroOrMore.

```
<payment>
  <transaction-id>#d9w72d83</transaction-id>
  <status>OK</status>
  <total currency="EUR">26.56</total>
  <datetime>2024-04-10T09:45:00</datetime>
  <order-items>
    <item qty="1"
          vat="10%"
          price="20.00"
          name="Cheese Cake 1kg"/>
    <item qty="1"
          vat="20%"
          price="3.80"
          name="Cement 20kg"/>
  </order-items>
</payment>
```

JSON Schema

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://interop.cs.univie/payment.schema.json",
  "title": "Payment Schema",
  "description": "Payment format for personal purchases.",

  "type": "object",
  "required": ["payment"],
  "properties": {
    "payment": {
      "type": "object",
      "required": [
        "transaction_id",
        "status",
        "total",
        "currency",
        "datetime",
        "order_items"
      ],
      "properties": {
        "transaction_id": { "type": "string" },
        "status":         { "enum": ["OK", "NOK"] },
        "total":          { "type": "number"},
        "currency":       { "enum": ["EUR"] },
        "datetime":       { "type": "string", "format": "date-time" },
        "order_items":    { "$ref": "#/$defs/order_items" }}}},

  "$defs": {
    "order_items": {
      "type": "array",
      "minItems": 1,
      "items": {
          "type": "object",
          "required": [
            "qty",
            "vat",
            "price",
            "name"
          ],
          "properties": {
            "qty":   { "type": "integer" },
            "vat":   { "type": "string", "pattern":"[0-9]+%" },
            "price": { "type": "number" },
            "name":  { "type": "string" }}}}}}
```

```
{
  "payment":
    {
      "transaction_id": "#d9w72d83",
      "status": "OK",
      "total": 26.56,
      "currency": "EUR",
      "datetime": "2024-04-10T09:45:00+00:00",
      "order_items": [
        { "qty": 1,
          "vat": "10%",
          "price": 20.00,
          "name": "Cheese Cake 1kg" },
        { "qty": 1,
          "vat": "10%",
          "price": 3.80,
          "name": "Cement 20kg" }
      ]
    }
}
```

JSON Schema

```
…
"$schema": "https://json-schema.org/draft/2020-12/schema",
"$id": "https://interop.cs.univie/payment.schema.json",
…
```

$schema specifies the specification version. *$id* defines a unique identifier for the schema.

```
{
  "payment":
    {
      "transaction_id": "#d9w72d83",
      "status": "OK",
      "total": 26.56,
      "currency": "EUR",
      "datetime": "2024-04-10T09:45:00+00:00",
      "order_items": [
        { "qty": 1,
          "vat": "10%",
          "price": 20.00,
          "name": "Cheese Cake 1kg" },
        { "qty": 1,
          "vat": "10%",
          "price": 3.80,
          "name": "Cement 20kg" }
      ]
    }
}
```

JSON Schema

```
  …
  "type": "object",
  "required": ["payment"],
  "properties": {
    "payment": {
                …
```

type either defines a structural element (array or object) or primitive types.

In this example we have *object* as the type. *properties* define all the keys that an object may contain. By default properties are not enforced if missing, this behaviour can be changed by adding the names of the properties to *required.*

```
{
  "payment":
    {
      "transaction_id": "#d9w72d83",
      "status": "OK",
      "total": 26.56,
      "currency": "EUR",
      "datetime": "2024-04-10T09:45:00+00:00",
      "order_items": [
        { "qty": 1,
          "vat": "10%",
          "price": 20.00,
          "name": "Cheese Cake 1kg" },
        { "qty": 1,
          "vat": "10%",
          "price": 3.80,
          "name": "Cement 20kg" }
      ]
    }
}
```

JSON Schema

```
…
"properties": {
   "status":         { "enum": ["OK", "NOK"]},
   "total":          { "type": "number"},
   "datetime":       { "type": "string",
                       "format": "date-time"},
   …
      "vat":   { "type": "string",
                 "pattern":"[0-9]+%" },
      …
```

```
{
  "payment":
    {
      "transaction_id": "#d9w72d83",
      "status": "OK",
      "total": 26.56,
      "currency": "EUR",
      "datetime": "2024-04-10T09:45:00+00:00",
      "order_items": [
        { "qty": 1,
          "vat": "10%",
          "price": 20.00,
          "name": "Cheese Cake 1kg" },
        { "qty": 1,
          "vat": "10%",
          "price": 3.80,
          "name": "Cement 20kg" }
      ]
    }
}
```

JSON Schema also provides a number of predefined data types.

For the type *string, format* enables to define predefined patterns for strings. See the official documentation for the available options.

JSON Schema

```
     …
     "properties": {
        …
        "order_items":    {
          "$ref": "#/$defs/order_items" }…},
        …
  …
  "$defs": {
    "order_items": {
      "type": "array",
      "minItems": 1,
      "items": {
          "type": "object",
          "properties": {…}}}}
          …
```

Properties can also reference internal and external JSON schema documents via *$ref*. In the example above the property *order_items* references the sub-schema defined in the local document under *$defs*.

```
{
  "payment":
    {
      "transaction_id": "#d9w72d83",
      "status": "OK",
      "total": 26.56,
      "currency": "EUR",
      "datetime": "2024-04-10T09:45:00+00:00",
      "order_items": [
        { "qty": 1,
          "vat": "10%",
          "price": 20.00,
          "name": "Cheese Cake 1kg" },
        { "qty": 1,
          "vat": "10%",
          "price": 3.80,
          "name": "Cement 20kg" }
      ]
    }
}
```

JSON Schema

```
    …
    "order_items": {
      "type": "array",
      "minItems": 1,
      "items": {
          "type": "object",
          "properties": {…}}}}
          …
```

The type array expects the property items which is expected to specify the structure of the items in the array. *minItems* specifies the minimum amount of items in the array expected.

```
{
  "payment":
    {
      "transaction_id": "#d9w72d83",
      "status": "OK",
      "total": 26.56,
      "currency": "EUR",
      "datetime": "2024-04-10T09:45:00+00:00",
      "order_items": [
        { "qty": 1,
          "vat": "10%",
          "price": 20.00,
          "name": "Cheese Cake 1kg" },
        { "qty": 1,
          "vat": "10%",
          "price": 3.80,
          "name": "Cement 20kg" }
      ]
    }
}
```

Exercises

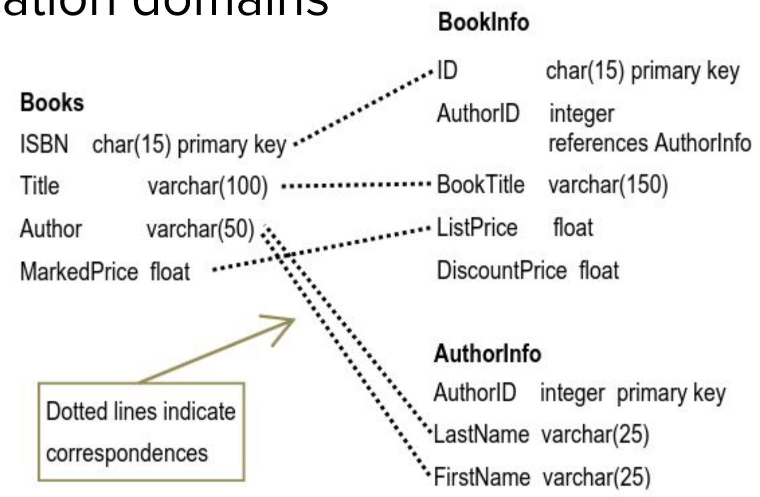
- Try to validate the data documents and schemas provided in the slides and experiment with the behaviour of the schema by making incremental changes to schema and data documents.

```
xmllint --noout --dtdvalid schema.dtd data.xml
xmllint --noout --schema schema.xsd data.xml
xmllint --noout --relaxng schema.rng data.xml
ajv validate --spec=draft2020 -c ajv-formats -s schema.json -d data.json
```

- For RelaxNG also use the following patterns: *optional*, *group*, *interleave*. See the tutorial for additional examples.
- Study the available data types for JSON here. Apply *oneOf, anyOf* and *formats*.

Schema Matching

- Goal of *match* operation (Rahm & Bernstein, 2001):
  - produce mappings between elements of two schemas,
  - which *semantically correspond* to each other.
- Relevant for many database-related application domains
- Typical examples include:
  - heterogeneous data integration
  - E-business
  - data warehousing
  - semantic query processing
- Use of (semi-)automatic schema matching when manual matching becomes infeasible.



Example from Bernstein et al. (2011)

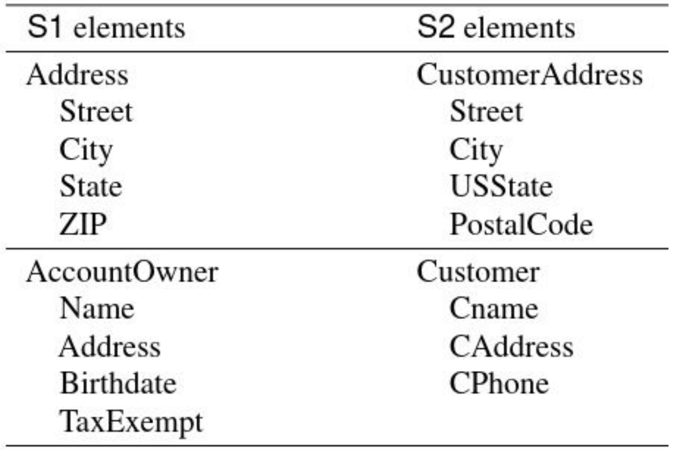
Model Management

Operations besides *match,* which usually is only one step of a multi-step data processing pipeline, (Bernstein et al. 2011):

- *merge* - unification of two schemas S1 and S2 into a single schema S3, also referred to as mediated schema
- *composing* - assuming a mapping between S1 and S2 (S1-S2) and S2 being altered into S2', then composing S1-S2 and S2-S2' yields mapping between S1-S2'
- *diff* - finds difference between mappings
- *extract* - returns complement of diff

Mapping Granularity

- A mapping represents a set of mapping elements. How the elements between S1 and S2 are related is defined by a *match expression.*
- *Element-level matching* maps elements in S1 with elements in S2. Typical example: atomic level match of a JSON object's property with a relation's column of a RDBS.
- *Structure-level matching* refers to matching combinations of elements appearing together in a structure.

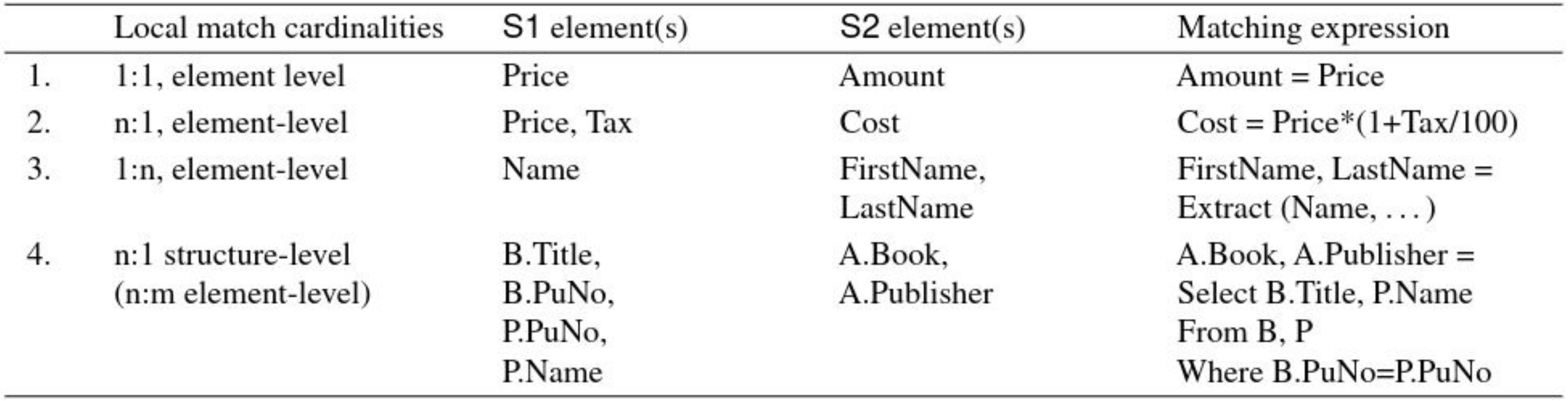


S1 elements	S2 elements
Address	CustomerAddress
Street	Street
City	City
State	USState
ZIP	PostalCode
AccountOwner	Customer
Name	Cname
Address	CAddress
Birthdate	CPhone
TaxExempt	

Top full, bottom partial structural match.
Example from Rahm & Bernstein (2001.)

Match Cardinalities

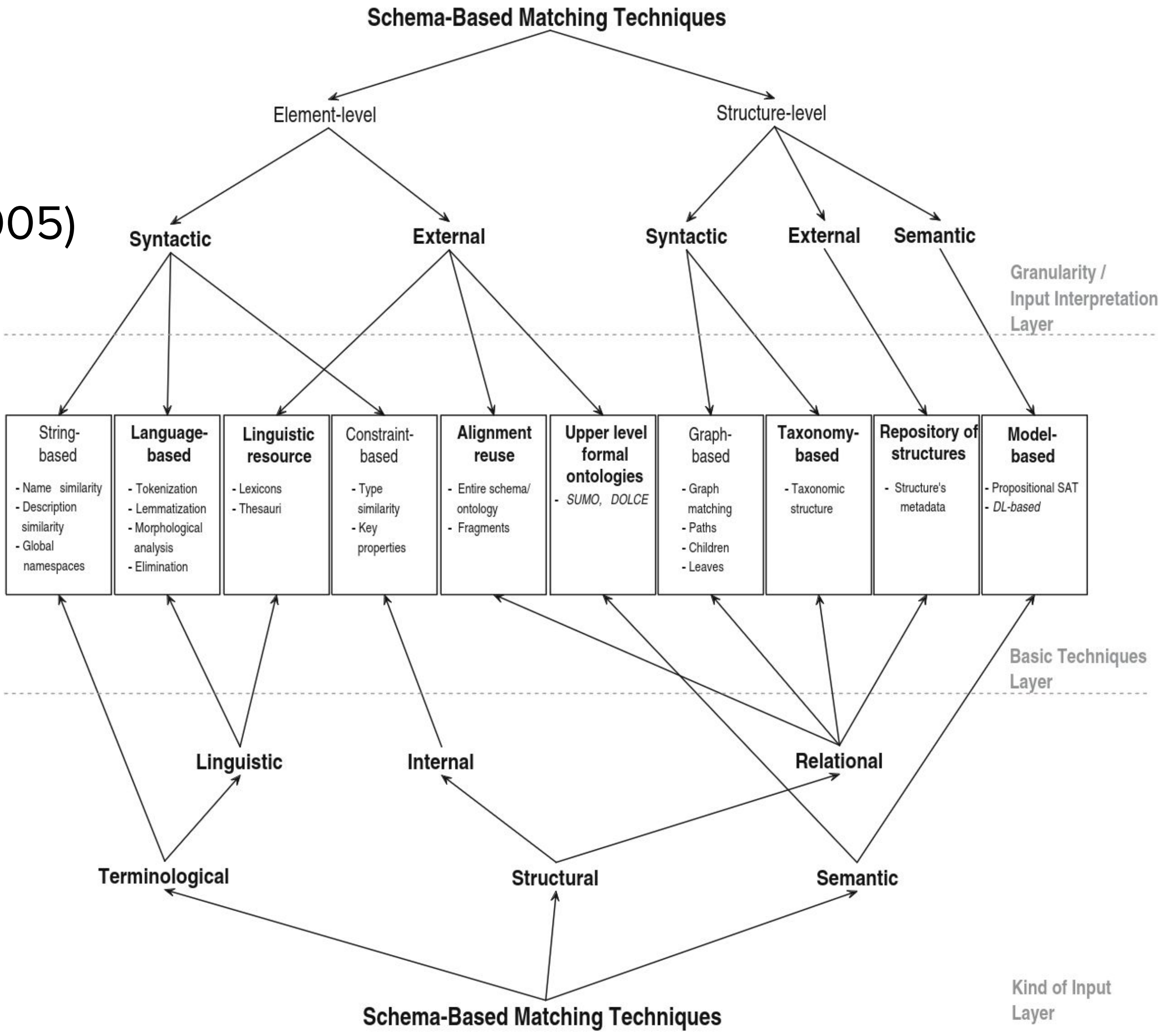
- An S1 or S2 element can participate in zero, one or many mappings
- Example (Rahm & Bernstein, 2001):



	Local match cardinalities	S1 element(s)	S2 element(s)	Matching expression
1.	1:1, element level	Price	Amount	Amount = Price
2.	n:1, element-level	Price, Tax	Cost	Cost = Price*(1+Tax/100)
3.	1:n, element-level	Name	FirstName, LastName	FirstName, LastName = Extract (Name, . . .)
4.	n:1 structure-level (n:m element-level)	B.Title, B.PuNo, P.PuNo, P.Name	A.Book, A.Publisher	A.Book, A.Publisher = Select B.Title, P.Name From B, P Where B.PuNo=P.PuNo

Approaches

Shvaiko & Euzenat (2005)



References

- Bernstein, P. A., Madhavan, J., & Rahm, E. (2011). Generic schema matching, ten years later. Proceedings of the VLDB Endowment, 4(11), 695-701.
- Rahm, E., & Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. the VLDB Journal, 10, 334-350
- Shvaiko, P., & Euzenat, J. (2005). A survey of schema-based matching approaches. In *Journal on data semantics IV* (pp. 146-171). Berlin, Heidelberg: Springer Berlin Heidelberg.
- Clark, J., & Makoto, M. (2001). RelaxNG Specification. https://relaxng.org/spec-20011203.html
- JSON Schema (2024). Specification. https://json-schema.org/specification