Database and Information Systems

Semi-Structured Data

- Metadata
- Semantic Web
- Web Services
- Dynamic Content 1: VoiceXML
- Dynamic Content 2: BPEL

Semi-Structured Data

Readings:

Antoniou, G. and van Harmelen, F.: A Semantic Web Primer. MIT Press, 2008.

W3C-recommendations

Semi-Structured Data

- required prerequisites
  - markup languages (LaTeX↓, HTML, SGML)
  - XML as metalanguage for defining markup languages
  - document type definition
    - content vs. attributes
    - identifier
  - namespaces
  - XML Schema
  - query languages: XPath, XQuery, SQL/XML

Semi-Structured Data

- Metadata
- Semantic Web
- Web Services
- Dynamic Content 1: VoiceXML
- Dynamic Content 2: BPEL

Metadata

- metadata: e.g. schema in a relational database
  - names of the relations
  - attributes of a relation
  - domain of each attribute
- metadata for documents
  - descriptive metadata: external to the content of the document
    - author
    - place and date of publication
    - length of the document
    - text genre
    - ...

Metadata

- metadata for documents (cont.)
  - Dublin Core Metadata Element Set: 15 attributes
    - Title, Creator, Subject, Description, Publisher, Contributor, Date, Type, Format, Identifier, Source, Language, Relation, Coverage, Rights
  - special purpose metadata
    - subject codes
    - key words taken from a thesaurus or ontology
    - e.g. MEDLINE: disease, anatomy, pharmaceuticals, ...
    - e.g. MARC: Machine readable cataloging record

Metadata

- RDF: resource description framework
  - goal: interoperability between applications
  - catalogue information
  - content rating
  - intellectual property rights
  - digital signatures for authentication
  - privacy levels
  - ...

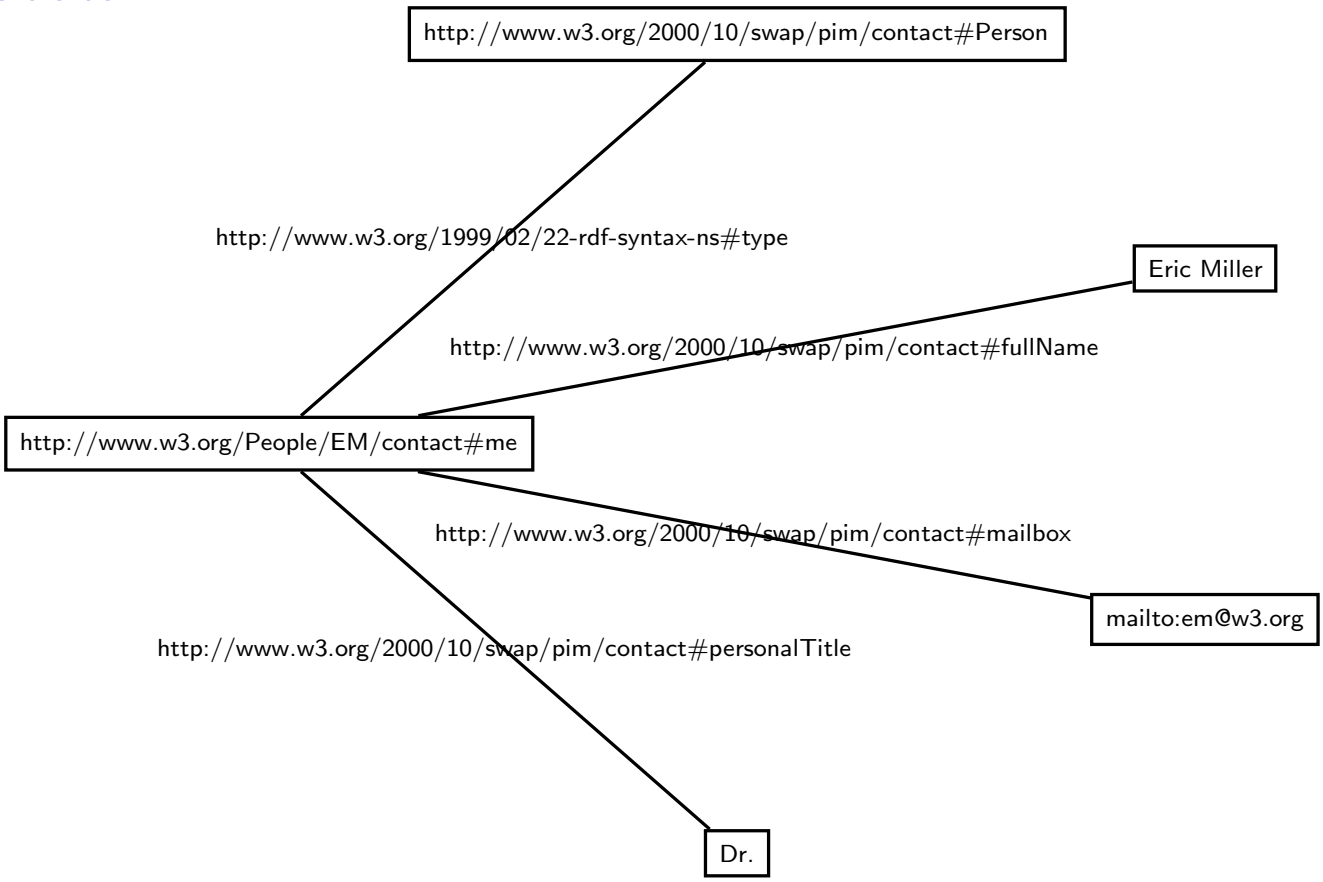
Metadata

- metadata for the author of a web-site

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact">
  <contact:Person
       rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Eric Miller</contact:fullName>
    <contact:mailbox rdf:resource="mailto:em@w3.org"/>
    <contact:personalTitle>Dr.</contact:personalTitle>
  </contact:Person>
</rdf:RDF>
```

- element, attribute and value specification by means of URIs (uniform resource identifiers)

Metadata



Metadata

- RDF: simple knowledge representation language
  - individuals
    - e.g. Eric_Miller
  - concepts (classes)
    - e.g. Person
  - properties of individuals
    - e.g. name, title, mailbox
  - values of properties
    - e.g. "Eric Miller", "Dr.", "em@w3.org"

Semi-Structured Data

Semantic Web

- enhancing web pages with rich semantic annotations to support automatic processing
  - retrieval
  - information extraction
  - translation
- providing web-based resources
  - ontologies
  - dictionaries

Semantic Web

- shortcomings of RDF
  - no data type definitions
  - no consistent expression for enumerations
- RDFS: RDF Schema:
  - type definitions of the XML schema (XSDL)
  - object-oriented knowledge representation
  - classes, subclasses
  - properties, data types
  - inheritance
  - individuals

Semantic Web

- DAML+OIL:
  DAML: DARPA agent markup language
  OIL: ontology interface layer
  - terminological reasoning
- revised into OWL: Web Ontology Language (2004)
  - relations between classes (e.g. disjointness)
  - cardinality restrictions (e.g. exactly one)
  - equality
  - types of properties
  - formal properties of properties (e.g. symmetry)
  - enumerated classes

RDF Schema

- class definitions

```
<rdfs:Class rdf:ID="Product">
  <rdfs:label>Product</rdfs:label>
  <rdfs:comment>An item sold by
        Super Sports Inc.</rdfs:comment>
</rdfs:Class>
```

- properties

```
<rdfs:Property rdf:ID="productNumber">
  <rdfs:label>Product Number</rdfs:label>
  <rdfs:domain rdf:resource="#Product"/>
  <rdfs:range rdf:resource=
    "http://www.w3.org/.../rdf-schema#Literal"/>
</rdfs:Property>
```

RDF Schema

- instances of classes:
  - defining resources to be of a particular RDF type
  - assigning properties

```
<Product rdf:ID="WaterBottle">
  <rdfs:label>Water Bottle</rdfs:label>
  <productNumber>38267</productNumber>
</Product>
```


RDF Schema

- subclasses:

```
<rdfs:Class rdf:ID="Outdoor">
  <rdfs:subClassOf rdf:resource="#Product">
  <rdfs:label>Outdoor Equipment</rdfs:label>
  <rdfs:comment>A subclass of products
        intended for outdoor activities
  </rdfs:comment>
</rdfs:Class>
```

OWL

- full terminological reasoning
- basis for intelligent web agents
- purpose
  - formalize a domain by defining classes and properties of those classes,
  - define individuals and assert properties about them, and
  - reason about these classes and individuals to the degree permitted by the formal semantics of the OWL language
- logical basis:
  - open world assumption
  - monotonic reasoning

OWL

- goal: being able to respond to queries like
  "Tell me what wines I should buy to serve with each course of the following menu. And, by the way, I don't like Sauternes."
- why?
  - special purpose XML languages support predefined transactions
  - but do not support general reasoning outside such a transaction
- added value:
  - giving justifications and explanations
  - switch between levels of granularity

OWL

- three sublanguages
  - OWL Lite
  - OWL DL
  - OWL Full

OWL

- OWL Lite
  - RDFS + `Class (Thing/Nothing)`, `Individual`, `equivalentClass`, `equivalentProperty`, `sameAs`, `differentFrom`, `AllDifferent`, `distinctMembers`, `Restriction`, `onProperty`, `allValuesFrom`, `someValuesFrom`, `minCardinality`, `MaxCardinality`, `cardinality`, `...`
  - simple applications of a class hierarchy with simple constraints only
    - limited cardinality restrictions (only for 0 and 1)
  - quick migration from existing resources (thesauri, taxonomies)

OWL

- OWL DL
  - maximum expressiveness while maintaining completeness and decidability
  - corresponds to description logics
  - classes cannot be instances of other classes
  - `oneOf`, `dataRange`, `disjointWith`, `unionOf`, `complementOf`, `intersectionOf`, arbitrary cardinality restrictions, ...

OWL

- OWL Full
  - maximum expressiveness without considering computability
  - e.g. class can be treated as a collection of individuals and an individual simultaneously
  - allows augmenting the semantics of the underlying RDF and OWL vocabulary

OWL

- upwards compatibility
  - Every legal OWL Lite ontology is a legal OWL DL ontology
  - Every legal OWL DL ontology is a legal OWL Full ontology
  - Every valid OWL Lite conclusion is a valid OWL DL conclusion
  - Every valid OWL DL conclusion is a valid OWL Full conclusion

OWL

- class and subclass definitions

```
<owl:Class rdf:ID="Winery"/>
<owl:Class rdf:ID="Region"/>
<owl:Class rdf:ID="ConsumableThing"/>

<owl:Class rdf:ID="PotableLiquid">
  <rdfs:subClassOf
        rdf:resource="#ConsumableThing" />
  ...
</owl:Class>
```

- instances of the class belong to the intersection of the restrictions

OWL

- class definition consists of name introduction or reference + list of restrictions

```
<owl:Class rdf:ID="Wine">
  <rdfs:label xml:lang="en">wine</rdfs:label>
  <rdfs:label xml:lang="fr">vin</rdfs:label>
  <rdfs:subClassOf
        rdf:resource="&food;PotableLiquid"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#madeFromGrape"/>
      <owl:minCardinality
        rdf:datatype="&xsd;nonNegativeInteger">
        1
      </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>
```


OWL

- individuals: three alternative definitions

```
<Region rdf:ID="CentralCoastRegion" />

<owl:Thing rdf:ID="CentralCoastRegion" />

<owl:Thing rdf:about="#CentralCoastRegion">
   <rdf:type rdf:resource="#Region"/>
</owl:Thing>
```

OWL

- properties

```
<owl:ObjectProperty rdf:ID="madeFromGrape">
  <rdfs:domain rdf:resource="#Wine"/>
  <rdfs:range rdf:resource="#WineGrape"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="course">
  <rdfs:domain rdf:resource="#Meal" />
  <rdfs:range rdf:resource="#MealCourse" />
</owl:ObjectProperty>
```

- properties are binary relationships:

  MadeFromGrapes(Wine,WineGrape)
  Course(Meal,MealCorse)

OWL

- properties can also be arranged in hierarchies

```
<owl:Class rdf:ID="WineDescriptor" />

<owl:Class rdf:ID="WineColor">
  <rdfs:subClassOf rdf:resource="#WineDescriptor" />
  ...
</owl:Class>

<owl:ObjectProperty rdf:ID="hasWineDescriptor">
  <rdfs:domain rdf:resource="#Wine" />
  <rdfs:range  rdf:resource="#WineDescriptor" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasColor">
  <rdfs:subPropertyOf
        rdf:resource="#hasWineDescriptor" />
  <rdfs:range rdf:resource="#WineColor" />
  ...
</owl:ObjectProperty>
```

OWL

- properties
  - relate individuals to individuals (object properties)
  - or individuals to datatypes (datatype properties)

OWL

- complex applications will need to refer to several ontologies
  → ontology mapping required
- equivalent classes

```
<owl:Class rdf:ID="TexasThings">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty
         rdf:resource="#locatedIn" />
      <owl:someValuesFrom
         rdf:resource="#TexasRegion" />
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```

OWL

- identity between individuals

```
<Wine rdf:ID="MikesFavoriteWine">
  <owl:sameAs
     rdf:resource="#StGenevieveTexasWhite" />
</Wine>
```


OWL

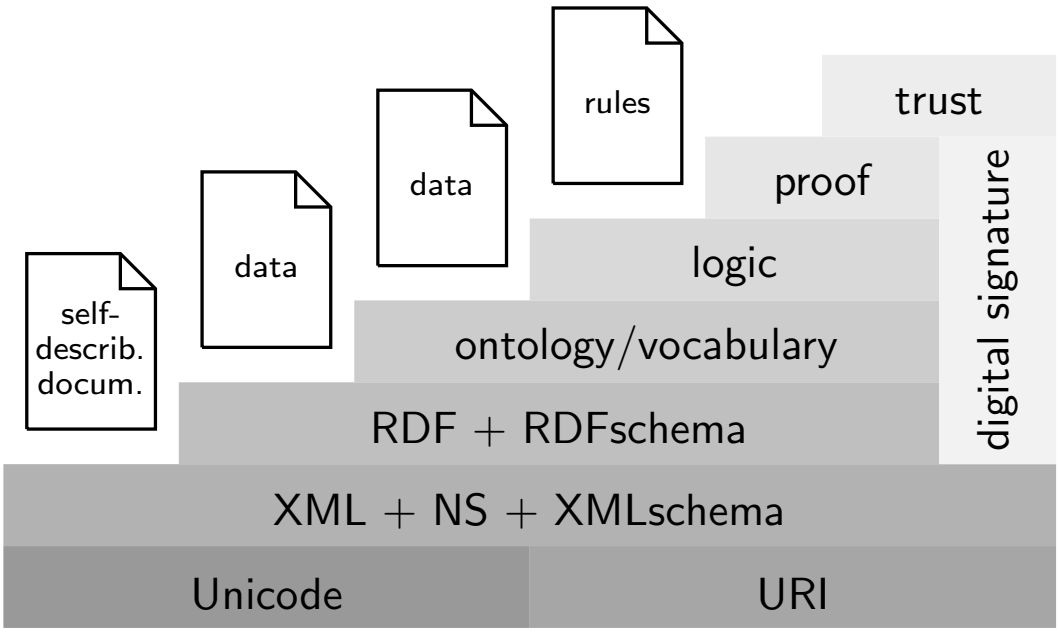
- different individuals

```
<WineSugar rdf:ID="Dry" />

<WineSugar rdf:ID="Sweet">
  <owl:differentFrom rdf:resource="#Dry"/>
</WineSugar>
<WineSugar rdf:ID="OffDry">
  <owl:differentFrom rdf:resource="#Dry"/>
  <owl:differentFrom rdf:resource="#Sweet"/>
</WineSugar>
```

Semantic Web

- a layered approach



original vision (Berners-Lee)

Semantic Web

- agents should be aware on which layer they are operating
- downward compatibility: agents aware of one layer should be able to interpret and use information on the lower layers
- upward partial understanding: agent should be able to take at least partial advantage of information on higher levels

- on the logic layer rule-based languages are considered as an alternative to OWL

Semi-Structured Data

- Metadata
- Semantic Web
- Web Services
- Dynamic Content 1: VoiceXML
- Dynamic Content 2: BPEL

Web Services

- fully automatic access to information systems over the web
  - hotel reservation
  - procurement
  - ...
- service needs to be self-explanatory
  - kind of service
  - access conditions (input information)
  - information/service provided (output information)
  - terms and conditions of usage

Web Services

- descriptions
  - message types
  - interface
  - bindings
  - access points
  - documentation (optional)

Web Services

```
<?xml version="1.0" encoding="utf-8" ?>
<description>
  <types> ... </types>
  <interface ... > ... </interface>
  <binding ... >   ... </binding>
  <service ... >   ... </service>
  <documentation>  ... </documentation>
</description>
```

Web Services

- declaration of message types

```
<types>
  <xs:schema
    xmlns:xs="http:www.w3.org/2001/XMLSchema" ...>
    <xs:element name="checkAvailability"
        type="tCheckAvailability"/>
    <xs:complexType name="tCheckAvailabilty">
      <xs:sequence>
        <xs:element name="checkInDate" type="xs:date" />
        <xs:element name="checkOutDate" type="xs:date" />
        <xs:element name="roomType" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
    <xs:element name="checkAvailabilityResponse"
        type="xs:double"/>
    <xs:element name="invalidDataError" type="xs:string"/>
  </xs:schema>
</types>
```


Web Services

- specification of the interface
  - abstract operations
  - messages to be received
  - messages to be sent to the client
  - interaction pattern: sequence of messages
    e.g. in-out pattern
    - services receives a message
    - answers with a reply message or a fault message

Web Services

- sample interface specification

```
<interface name="reservationInterface">
  <fault name="invalidDataFault"
      element="gnhs:invalidDataError" />
  <operation name="opCheckAvailability"
      pattern="www.w3.org/ns/wsdl/in-out"
      style="www.w3.org/ns/wsdl/style/iri"
      wsdlx:safe="true">
    <input messageLabel="In"
        element="gnhs:checkAvailability" />
    <output messageLabel="Out"
        element="gnhs:checkAvailabilityResponse" />
    <outfault ref="tns:invalidDataFault" messageLabel="Out" />
  </operation>
</interface>
```


Web Services

- bindings: how to exchange a message
  - message format
  - transmission protocol (http, soap, ...)

  for operations and faults
- access points
  - interface
  - list of endpoints (binding and address)

Web Services

- message exchange patterns
  - inbound patterns /outbound patterns
  - in-only pattern
    - exactly one inbound message
    - no fault propagation possible
  - robust in-only pattern
    - incoming message might trigger the return of a fault message
  - in-out pattern
    - an incoming message followed by a reply to the originator
    - the reply can by a fault message
  - out-only, robust-out-only, in-optional-out, out-in, out-optional-in

Web Services

- message exchange pattern provide a minimal contract between the partners
  - define only placeholder for messages
  - abstract specifications
    - message types and binding information needs to be specified in the definition of an `operation`
  - do not describe an interaction exhaustively
    - additional message exchanges can be specified
  - identified by URIs not by names
    - extensibility: additional pattern can be defined

Web Services

- WSDL specifications can be translated to RDF
  - all WSDL components (interfaces, operations, bindings, services, endpoints, ... are mapped to resources (identified by a URI)
  - all these resources receive an appropriate RDF type
  - all WSDL relationships are translated into RDF statements with appropriate properties,
    - operation belongs to an interface
    - operation has a certain interaction style

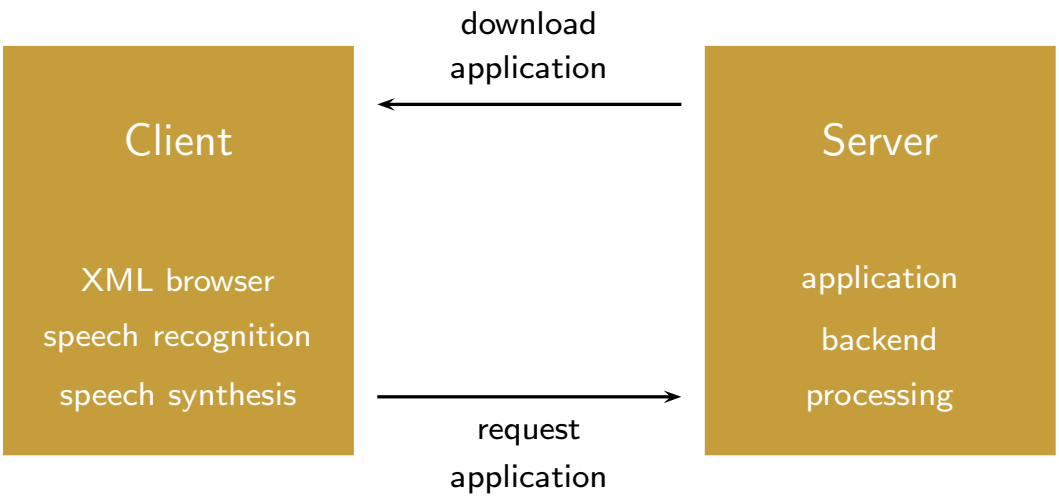
Semi-Structured Data

VoiceXML

- markup Languages are not restricted to static data
- can receive a procedural semantics
- example: VoiceXML
  - interactive applications with spoken language
- spoken language is always strictly sequential
  - requires careful design of the human-computer interface

VoiceXML

- dialogue descriptions in VoiceXML
  - are downloaded from the server to the client
  - control the speech synthesiser/recogniser at the client



VoiceXML

- acoustic menue

```
<?xml version="1.0" encoding= ...>
<vxml version="2.0" lang="en">
<menu>
  <prompt> Say one of: <enumerate/> </prompt>
  <choice next="http://...sports.vxml">
     Sports </choice>
  <choice next="http://...stocks.vxml">
     Stocks </choice>
  <choice next="http://...news.vxml">
     News </choice>
  <noinput> Please say one of <enumerate/> </noinput>
</menu>
</vxml>
```


VoiceXML

- forms

```
<form>
  <field name="city">
    <prompt> Where do you want to travel to? </prompt>
    <option> Berlin </option>
    <option> London </option>
    ...
  </field>
  <field name="number" type="number">
    <prompt> How many persons will travel to
             <value expr="city"/> ? </prompt>
  </field>
  <block> <submit next="http://...handler.html"
      namelist="city number"/> </block>
</form>
```

VoiceXML

- dynamic prompts (iteration)

```
<form>
  <field name="number" type="number">
    <prompt count=1> How many persons will
        travel to <value expr="city"/> ? </prompt>
    <prompt count=2> Please tell me the number
        of persons travelling. </prompt>
    <prompt count=3> To book a flight, you must tell
        me the number of people travelling to
        <value expr="city"/> ? </prompt>
    <nomatch>
      <prompt> Please say just a number. </prompt>
      <reprompt/>
    </nomatch>
  </field>
</form>
```


VoiceXML

- arithmetic operations, integrity checks (conditionals)

```
<form>
  <field name="number" type="number">
    <prompt> How many persons will
         travel to <value expr="city"/> ? </prompt>
    <filled>
      <var name="num_trav" expr="number + 0"/>
      <if cond="num_trav > 10">
        <prompt> Sorry, we only handle groups of up
          to 10 people. </prompt>
        <clear namelist="number"/>
      </if>
    </filled>
  </field>
</form>
```

Semi-Structured Data

- Metadata
- Semantic Web
- Web Services
- Dynamic Content 1: VoiceXML
- Dynamic Content 2: BPEL

BPEL

- business process execution language
- orchestration of web services
- centralized approach (conductor!)
  - invoking services
  - processing their results
  - coordination of asynchroneous communication between services
  - correlating message exchanges between participants
  - implementing parallel processing activities
  - manipulating data between interactions
  - supporting long running business transactions and activities
  - consistent exception handling

BPEL

- description of complex workflows
  - coordinating a hotel reservation and a flight ticket
    - get a number of alternative offers from different vendors
    - find the optimal combination under given constraints
    - make the reservations
  - remote maintenance procedures
    - find out the individual configuration
    - choose the corresponding upload
    - upload the update
- steps in a workflow can fail $\longrightarrow$ fault handling

BPEL

- describing *processes* consisting of *activities*
  - process descriptions are executed by a orchestration engine
  - orchestration engine has access to the local or remote services
- basic activities / structured activities
- abstract processes / executable processes
- web services involved in a process are defined as `partnerLink` with
  - a `name`,
  - a `partnerLinkType`, and
  - role specifications (`myRole`, `partnerRole`)
- variables can be set to a value and accessed
- variables are scoped according to the XML structure
- processes can be initiated by a `receive` or `pick` activity with attribute `createInstance="yes"`

BPEL

- basic activities
  - `invoke`: invoking a service with attributes `partnerLink`, `portType`, `operation`, `inputVariable`, and `outputVariable`
  - `receive`: receiving a message with attributes `partnerLink`, `portType`, `operation`, `variable`, and optional `createInstance`
  - `reply`: replying to a message with attributes `partnerLink`, `portType`, `operation`, `variable`, and `faultName`
  - signaling faults
  - waiting until a deadline or for a certain period of time
  - doing nothing (`empty`)

BPEL

- structured activities
  - `sequence`: one or more activities processed sequentially
  - `switch`: conditional branching

```
<switch>
  <case condition="..."> ... </case>
  <otherwise> ... </otherwise>
</switch>
```

  - `while`: iteration

```
<while condition="..."> ... </while>
```

  - `pick`: waits until an event occurs and then processes the associated activity
    - needs at least one `onMessage` element with attrributes `partnerLink`, `portType`, `operation`, and `variable`
    - `onMessage` works thes same as `receive` (i.e. allows tro specify a `reply`

BPEL

- structured activities (cont.)
  - `flow`: execute several activities in parallel
    - synchronization of activities is possible
  - correlation: making sure that the partner is always the same instance of a service by using identifying variables
  - compensation handler, fault handler (`catch`), event handler
    - all handlers are scoped