

**X
M
M
M**

Service Oriented Architecture: (Semantic) Web Services, (Business) Process Modeling, Software Engineering

- Lecture Notes -

[Yuhong Yan](#)

[Harold Boley](#)

[Bruce Spencer](#)

NRC-IIT Fredericton

Internet Logic

OWL

RuleML

jDREW

UDDI

WSDL

SOAP

BPEL

[ICEC 2006 Tutorial](#)

13 Aug 2006

Agenda

- Introduction (20 minutes)
 - Service and Service oriented architecture
 - XML
- Web Services (30 minutes)
 - Web Services as middleware
- Formal Methods for Web Services Process Modeling (30 minutes)
 - Automata, Process Algebra and Petri Nets
- Break
- Semantic Web Services (75 minutes)
 - RDF, Description Logic, OWL, RuleML, OWL-S, WSMO
- Wrap-up (10 minutes)

Introduction to Service and Service Oriented Architecture

Intro

Intro

Service and Service Science

- Service (from IBM):

A service is a provider/client interaction that creates and captures value.

- Service sector is important in post-manufacturing countries
 - 80% of economic activities in US (from National Academy of Engineering 2003)
 - 70% of the Organization for Economic Cooperation and Development (OECD) countries
- Service Science
 - An new emerging discipline in business schools
 - Marketing, customer relation, operations research, business, macroeconomics

Service Computing

- The role of Computing in Services
 - Facilitating the communication, storage, and processing of information
 - Decreasing the cost of customization and marketing to small segments
 - Giving the customer a broader range of selection
- In this tutorial
 - Automated services enabled by Internet techniques
 - Communicating among services
 - Composing services into a process
 - Semantic-based service searching and matching

Service Oriented Architecture (SOA)

- *The OASIS SOA Reference Model group defines **Service Oriented Architecture** is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.*

Web Services

- W3C Web Services Architecture:

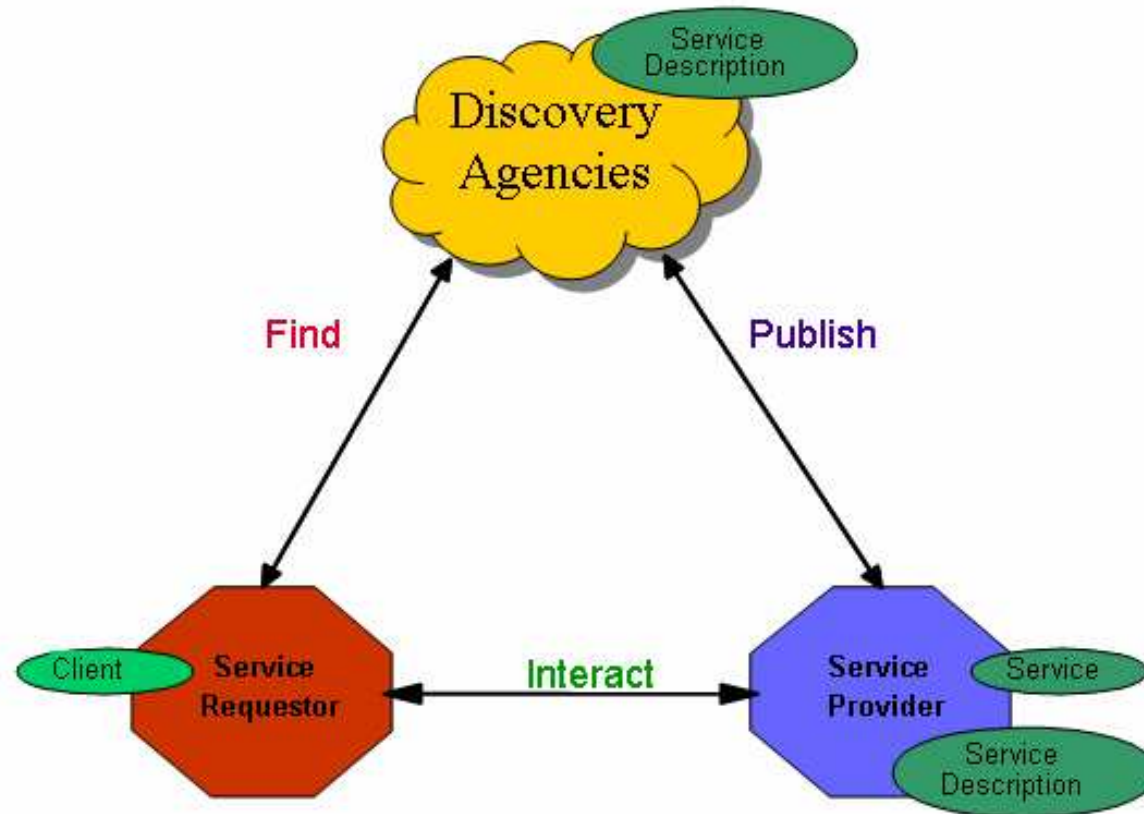
*“A **Web service** is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format.”*



Web Services vs. SOA

- Two opinions:
 - SOA typically refers to Web Services –W3C
 - Web service **reference** architecture is on the foundation of SOAP and WSDL
 - SOA is not the same as Web Services (in this tutorial too)
 - Web services are an instantiation of SOA with SOAP and WSDL
 - SOA is a concept not bound to any specific technology
- What people agree
 - The roles and operations in the SOA/WS triangle
 - The principles of SOA/WS
 - There are many ways to implement messaging and service description language, but ought to use internet protocols

SOA/Web Service triangle



From "Web Services Architecture W3C Working Draft"
<http://www.w3.org/TR/2002/WD-ws-arch-20021114/>

SOA/WS Principles

- **Service Encapsulation**
- **Service Loose coupling** - Services maintain a relationship that minimizes dependencies and only requires that they maintain an awareness of each other
- **Service contract** - Services adhere to a communications agreement, as defined collectively by one or more service description documents
- **Service abstraction** - Beyond what is described in the service contract, services hide logic from the outside world
- **Service reusability** - Logic is divided into services with the intention of promoting reuse
- **Service composability** - Collections of services can be coordinated and assembled to form composite services
- **Service autonomy** – Services have control over the logic they encapsulate
- **Service statelessness** – Services minimize retaining information specific to an activity
- **Service discoverability** – Services are designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanism

http://en.wikipedia.org/wiki/Service-oriented_architecture

Extensible Markup Language



General Advantages of XML

XML offers new general possibilities, from which SOA can profit:

- (1) Definition of self-describing data in worldwide standardized, non-proprietary format
- (2) Structured data and knowledge exchange for enterprises in various industries
- (3) Integration of information from different sources (into uniform documents)



Address Example: External to HTML

External Presentation:

Xaver M. Linde
Wikingerufer 7
10555 Berlin

*HTML tags are still
presentation-oriented*

HTML Markup:

```
<em>Xaver M. Linde</em>  
<br>  
Wikingerufer 7  
<br>  
<strong>10555 Berlin</strong>
```



Address Example: HTML to XML

HTML Markup:

```
<em>Xaver M. Linde</em>  
<br>  
Wikingerufer 7  
<br>  
<strong>10555 Berlin</strong>
```

While not conveying
any formal semantics:

*XML tags are chosen for
content-structuring needs*

XML Markup:

```
<address>  
  <name>Xaver M. Linde</name>  
  <street>Wikingerufer 7</street>  
  <town>10555 Berlin</town>  
</address>
```

Address Example: XML to External

XML Markup:

```
<address>  
  <name>Xaver M. Linde</name>  
  <street>Wikingerufer 7</street>  
  <town>10555 Berlin</town>  
</address>
```

*XML stylesheets are,
e.g., usable to generate
different presentations*

External Presentations:

Xaver M. Linde
Wikingerufer 7
10555 Berlin

...

Xaver M. Linde
Wikingerufer 7
10555 Berlin



Address Example: XML to XML

XML Markup 1:

```
<address>  
  <name>Xaver M. Linde</name>  
  <street>Wikingerufer 7</street>  
  <town>10555 Berlin</town>  
</address>
```

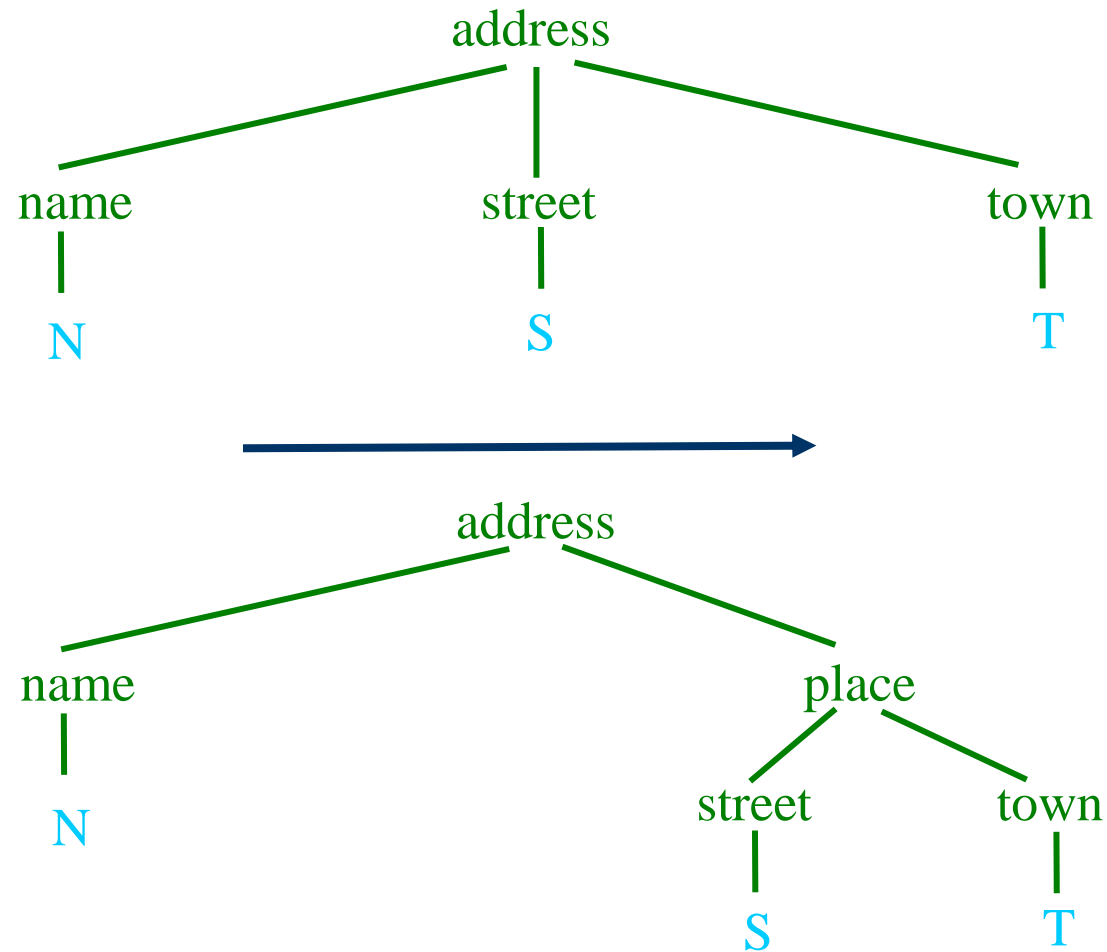
XML Markup 2:

```
<address>  
  <name>Xaver M. Linde</name>  
  <place>  
    <street>Wikingerufer 7</street>  
    <town>10555 Berlin</town>  
  </place>  
</address>
```

XML stylesheets are also usable to transform XML representations



Address Example: Some Stylesheets Will Contain Term-(Tree-)Rewriting Rules



X

M

L

Address Example: The Element Tree

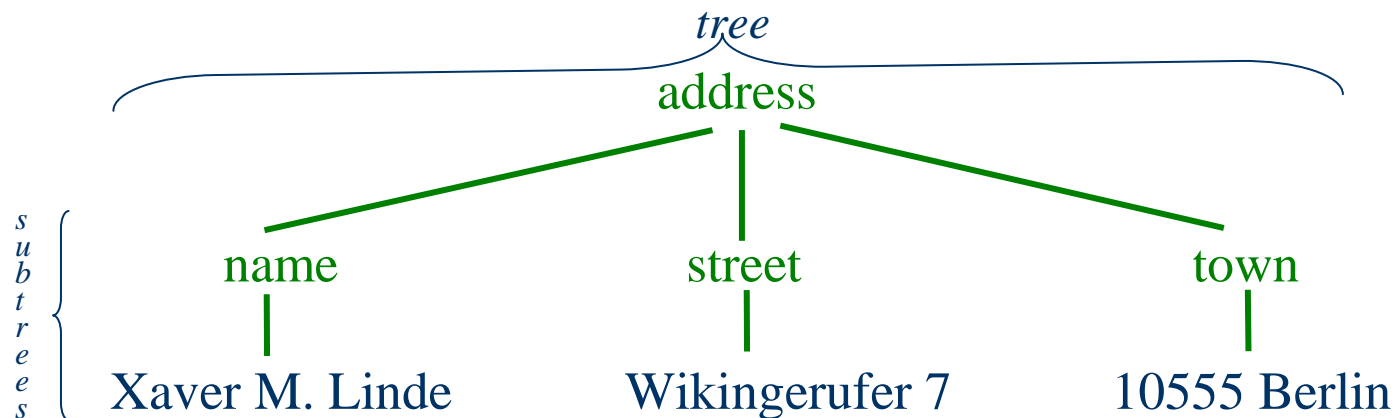
XML Markup:

e
l
e
m
e
n
t { `<address>` *subelements*
 `<name>Xaver M. Linde</name>`
 `<street>Wikingerufer 7</street>`
 `<town>10555 Berlin</town>`
 `</address>`

Prolog Term:

s
t
r
u
c
t
u
r
e { `address(` *substructures*
 `name("Xaver M. Linde"),`
 `street("Wikingerufer 7"),`
 `town("10555 Berlin")`
 `)`

Node-Labeled, (Left-to-Right-)Ordered Element Tree:



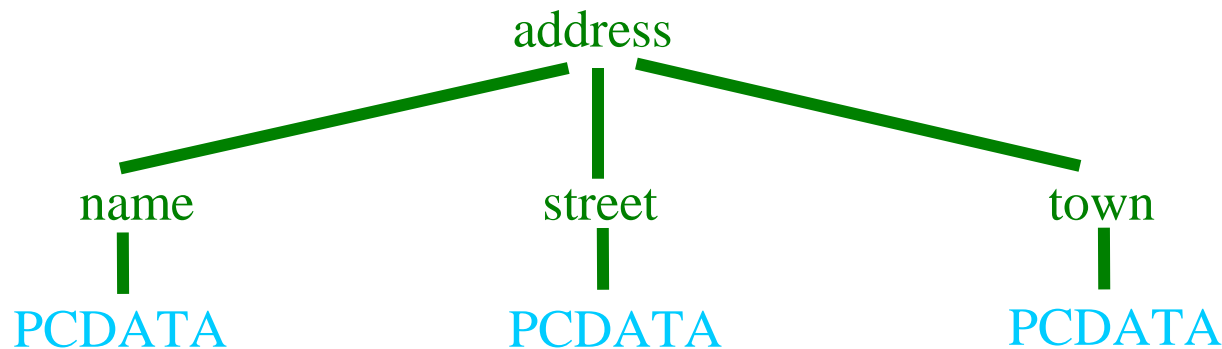
Address Example: Document Type Definition and Tree (1)

Document Type Definition (DTD):

Extended Backus-Naur Form (EBNF):

<!ELEMENT address	(name, street, town) >	address ::= name street town
<!ELEMENT name	(#PCDATA) >	name ::= PCDATA
<!ELEMENT street	(#PCDATA) >	street ::= PCDATA
<!ELEMENT town	(#PCDATA) >	town ::= PCDATA

Document Type Tree:

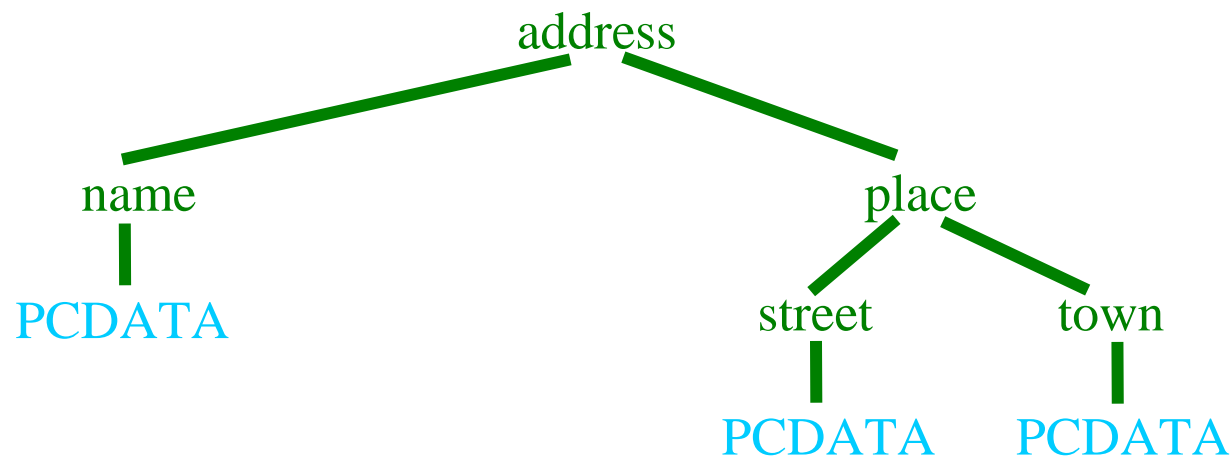


Address Example: Document Type Definition and Tree (2)

Document Type Definition (DTD):

```
<!ELEMENT address      (name, place) >  
<!ELEMENT place        (street, town) >  
<!ELEMENT name         (#PCDATA) >  
<!ELEMENT street       (#PCDATA) >  
<!ELEMENT town         (#PCDATA) >
```

Document Type Tree:



Well-Formedness and Validity

XML principles for a document being *well-formed*:

- Open and close all tags
- Empty tags end with />
- There is a unique root element
- Elements may not overlap
- Attribute values are quoted
- < and & are only used to start tags and entities
- Only the five predefined entity references are used

XML principle for a document being *valid* with respect to (w.r.t.) a DTD :

- Match the constraints listed in the DTD (or, generate from DTD as linearized derivation tree, as shown later)

Checked by **validators** such as

<http://www.stg.brown.edu/service/xmlvalid/>



Mail-Box Example: Address Variant

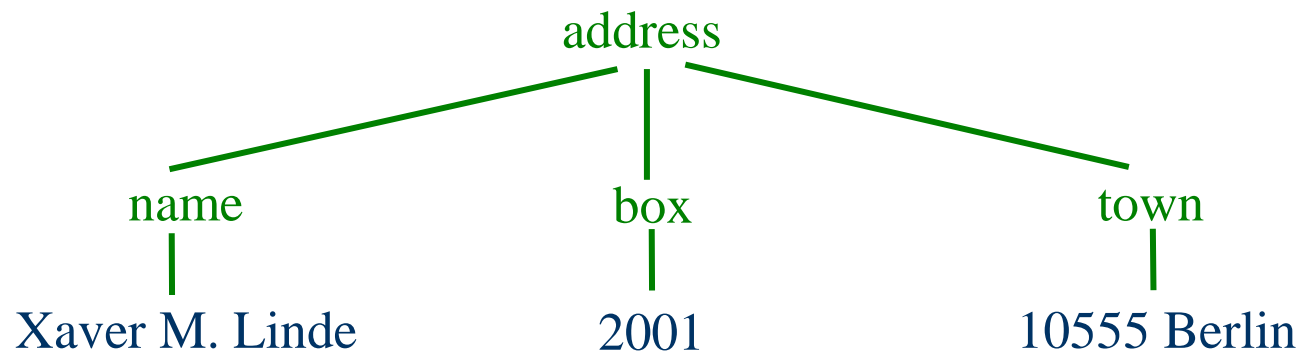
XML Markup:

```
<address>
  <name>Xaver M. Linde</name>
  <box>2001</box>
  <town>10555 Berlin</town>
</address>
```

Prolog Term:

```
address(
  name("Xaver M. Linde"),
  box("2001"),
  town("10555 Berlin")
)
```

Node-Labeled, (Left-to-Right-)Ordered Element Tree:



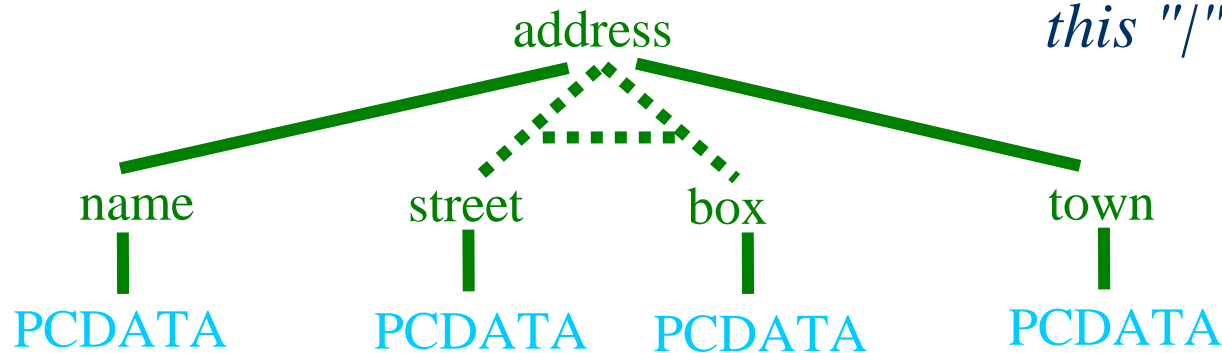
"|" - Disjoined Street/Mail-Box Example: Document Type Definition and Tree

Document Type Definition (DTD):

```
<!ELEMENT address      (name, (street | box), town) >
<!ELEMENT name         (#PCDATA) >
<!ELEMENT street       (#PCDATA) >
<!ELEMENT box          (#PCDATA) >
<!ELEMENT town         (#PCDATA) >
```

"/": Choice
The above box address and the original street address are valid w.r.t. this "/"-DTD

Document Type Tree:



Phone & Fax Example: Address Variant

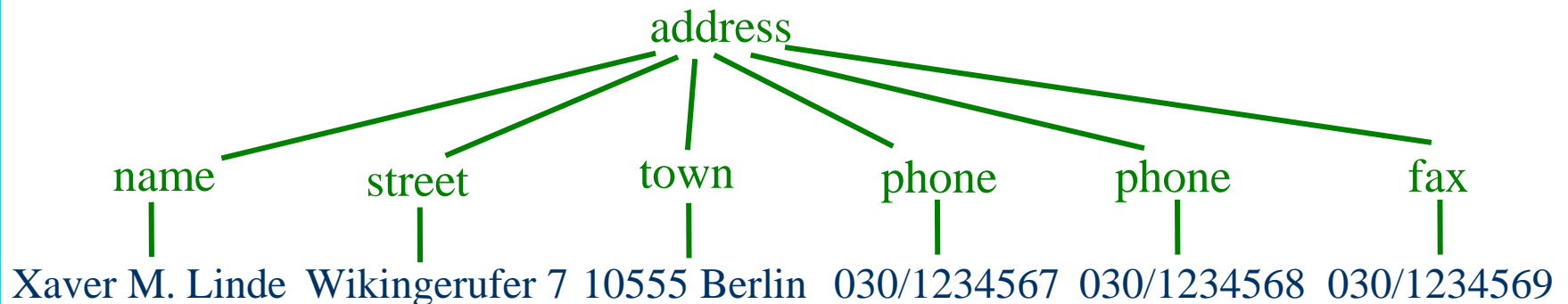
XML Markup:

```
<address>
  <name>Xaver M. Linde</name>
  <street>Wikingerufer 7</street>
  <town>10555 Berlin</town>
  <phone>030/1234567</phone>
  <phone>030/1234568</phone>
  <fax>030/1234569</fax>
</address>
```

Prolog Term:

```
address(
  name("Xaver M. Linde"),
  street("Wikingerufer 7"),
  town("10555 Berlin"),
  phone("030/1234567"),
  phone("030/1234568"),
  fax("030/1234569")
)
```

Node-Labeled, (Left-to-Right-)Ordered Element Tree:



"+"/"*" -Repetitive-Phone & -Fax Example: Document Type Definition and Tree

Document Type Definition (DTD):

<!ELEMENT address (name, street, town, phone+, fax*) >

<!ELEMENT name (#PCDATA) >

<!ELEMENT street (#PCDATA) >

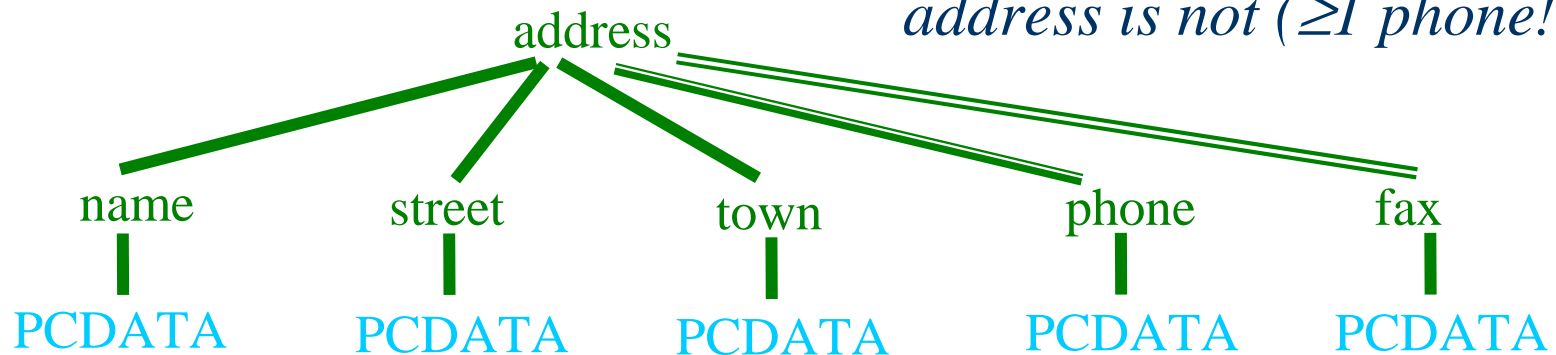
<!ELEMENT town (#PCDATA) >

<!ELEMENT phone (#PCDATA) >

<!ELEMENT fax (#PCDATA) >

"+"/"" : One/Zero or More
The above two-phone/one-fax
address is valid w.r.t. this
"+"/"*" -DTD but the
original no-phone/no-fax
address is not (≥ 1 phone!)*

Document Type Tree:



Country Example: Address Variant

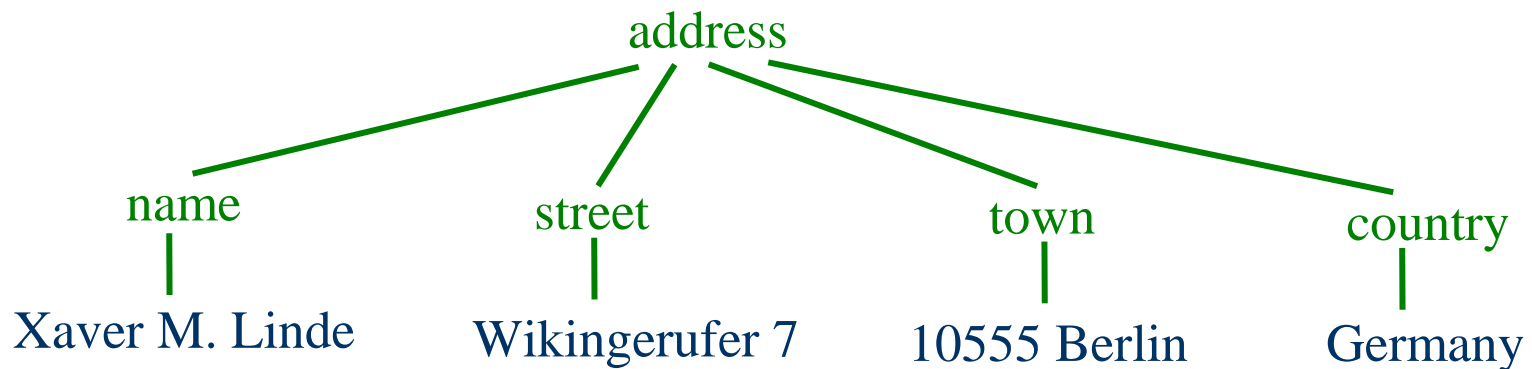
XML Markup:

```
<address>
  <name>Xaver M. Linde</name>
  <street>Wikingerufer 7</street>
  <town>10555 Berlin</town>
  <country>Germany</country>
</address>
```

Prolog Term:

```
address(
  name("Xaver M. Linde"),
  street("Wikingerufer 7"),
  town("10555 Berlin"),
  country("Germany")
)
```

Node-Labeled, (Left-to-Right-)Ordered Element Tree:



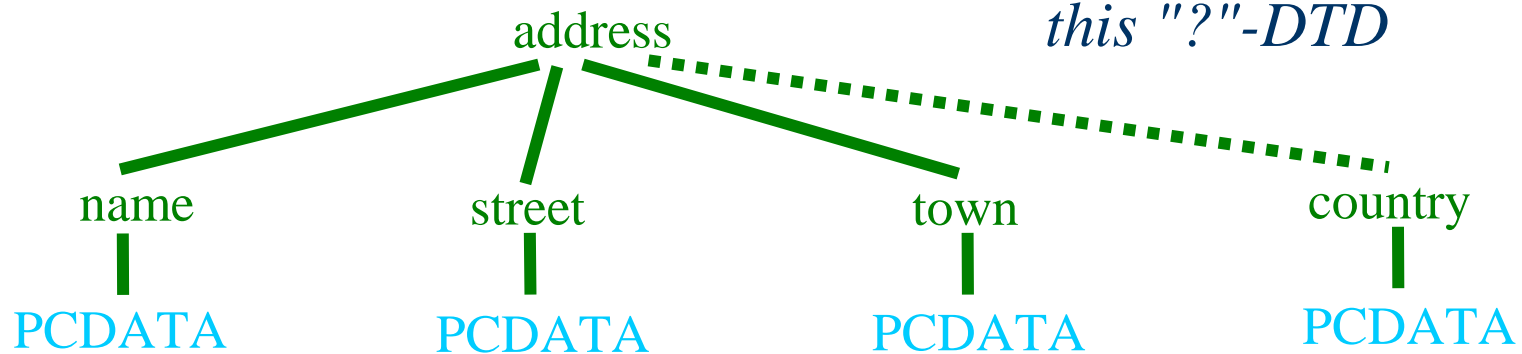
"?"-Optional-Country Example: Document Type Definition and Tree

Document Type Definition (DTD):

```
<!ELEMENT address (name, street, town, country?) >  
<!ELEMENT name (#PCDATA) >  
<!ELEMENT street (#PCDATA) >  
<!ELEMENT town (#PCDATA) >  
<!ELEMENT country (#PCDATA) >
```

*"?": One or Zero
The above country
address and the
original countriless
address are valid w.r.t.
this "?"-DTD*

Document Type Tree:



Country Address: A Complete XML Document Referring to an External DTD

XML Document (just ASCII, e.g. stored in a file):

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE address SYSTEM "country-address.dtd">
<address>
  <name>Xaver M. Linde</name>
  <street>Wikingerufer 7</street>
  <town>10555 Berlin</town>
  <country>Germany</country>
</address>
```

The *XML declaration* uses standalone attribute with "no" value: DTD import

The *DOCUMENT TYPE declaration* names the *root element* address and, after the *SYSTEM keyword*, refers to an *external DTD* "country-address.dtd"

(or, at some absolute URL, to an "http://www.test.org/country-address.dtd")



XML Namespaces

Namespaces

2007-2

XML Namespaces and Programming-Language Modules

- XML namespaces are akin to namespaces, packages, and modules in programming languages
- Disambiguation of tag–and attribute–names from different XML applications (“spaces”) through different prefixes
- A *prefix* is separated from the local *name* by a “:”, obtaining *prefix:name* tags
- Namespaces constitute a layer on top of XML 1.0, since *prefix:name* is again a valid tag name and namespace bindings are ignored by some tools

Namespace Bindings

- Prefixes are bound to namespace URIs by attaching an `xmlns:prefix` attribute to the prefixed element or one of its ancestors, `prefix:name1`, ..., `prefix:namen`
- The value of the `xmlns:prefix` attribute is a URI, which may or (unlike for DTDs!) may not point to a description of the namespace's syntax
- An element can use bindings for multiple namespaces via attributes `xmlns:prefix1`, ..., `xmlns:prefixm`

Namespaceless Example: Address Variant

Namespaceless XML Markup:

```
<address>  
  <name>Xaver M. Linde</name>  
  <street>Wikingenufer 7</street>  
  <town>10555 Berlin</town>  
  <bill>12.50</bill>  
  <phone>030/1234567</phone>  
  <phone>030/1234568</phone>  
  <fax>030/1234569</fax>  
  <bill>76.20</bill>  
</ address>
```

*bill is ambiguous
tag (name clash
from two XML
applications)*

Two-Namespace Example: Snail-Mail and Telecoms Address Parts

Namespace XML Markup:

```
<mail:address xmlns:mail="http://www.deutschepost.de/"
              xmlns:tele="http://www.telekom.de/">
  <mail:name>Xaver M. Linde</mail:name>
  <mail:street>Wikingerufer 7</mail:street>
  <mail:town>10555 Berlin</mail:town>
  <mail:bill>12.50</mail:bill>
  <tele:phone>030/1234567</tele:phone>
  <tele:phone>030/1234568</tele:phone>
  <tele:fax>030/1234569</tele:fax>
  <tele:bill>76.20</tele:bill>
</ mail:address>
```

*bill disambiguation
through mail and
tele prefixes*

- The root element, **mail:address**, as well as the children **mail:name**, **mail:street**, **mail:town**, and **mail:bill**, use the **mail** prefix, bound to a deutschepost URI
- The **tele:phone**, **tele:fax**, and **tele:bill** children use the **tele** prefix, bound to a telekom URI