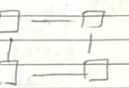
Page No.
Date

Environment → centralized
→ Distributed

Centralized

Fully on mainframe

Distributed



Storage of individual is much less than that of mainframe. But connected together to obtain high storage capacity and computational power

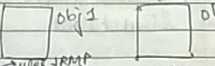
Why distributed env?
→ Cost effective (Super computers are expensive)
→ Break up a task into multiple parts to decrease the computational time without the use of mainframe

SIMD - Single Instruction Multiple Data

Socket prog. — OS level API
- More control over socket behaviour
- Relatively complex when compared to prog. level API

consists of IP addr and port no. of client & server

Distributed Objects



∴ CORBA is used (IDL is used)

platform independent but not prog. lang independent

(RMI can be used)

Both the obj need to be coded in Java

JAMR & IIOP are heavy weight protocol and thus, web services use SOAP. It is based on XML

9/7/19 XML - Service Oriented computing

Extensible (Create your own datatype)
- allows user to specify what each tag and attribute means

additional info attached with the text to make its structure comprehensible

XML	HTML
- Presentation & content are separated. Hence, parser is easier to create	- Parser difficult to create
- Can mix different types of data	- No data validation as it is predefined data structure. It cannot accomodate any other data which does not fit the data structure.

⇒ Base of both HTML & XML is SGML

Features of XML:
i) Extensibility
ii) Separation of data & presentation
iii) Structured
iv) Validation

Components of XML: i) XML document (data resides here)
abc.i = 10

struct abc{ int i; int j; };
main(){ abc a;}

↑ → Schema doc. (definition & create instance)

ii) DTD (Document Type Declaration)

iii) XSL (Extensible Style Sheet Lang.)

XML doc:

```
<students>
   <student>
      <id> 1 </id>
      <name>
         <fname> abc </fname>
         <lname> xyz </lname>
      </name>
   </student>
   :
</students>
```

⇒ Ele. have opening & closing tag. unless it is an empty ele.

⇒ Attribute values are quoted.

⇒ No isolated mark-up char. Ex: < >

⇒ Tags can be: - Semantic data representation
- Business rules

⇒ Tags can be used across domains. (Ex- Healthcare & insurance, Banking)

XML Validation: XML data is constrained by rules. If the received data follows these rules, it is considered a valid XML doc.

⇒ A XML doc. ~~can~~ may be well-formed but, not valid.

XML Schema

Defines: · Structure
· Syntax Vocabulary
] Syntax

⇒ XML doc. has to be well-formed (proper opening & closing tag)
XML Schema checks validity (if XML doc. satisfies the schema it is well-formed & valid doc.)

⇒ Comments : <!-- -->

⇒ namespace is analogous to packages in Java (imported namespace ; target namespace)
declared using xmlns : name = value
provide uniqueness to your elements.
URI recommended

⇒ Parsers :
i) DOM
ii) SAX

⇒ Before parsing, the XML doc. is converted into a tree-like structure and this is called XML doc. model (processing)

10/7/19 XML Schema : Definition & declaration
↑ create new types
↑ Enable ele with diff. names & types.

<!-- definition -->
<xsd : simpleType name = "TitleType">
prefix for some namespace.
↑ can be complex.
↑ some attribute tag / ele to be defined

String but with some restriction

Page No.
Date

Datatype

<xsd: restriction base = "xsd: string" >
<xsd: enumeration value = "Mr." />
"Prof."
"Dr."

</xsd: restriction>
</xsd: simpleType>

<!- Declaration ->
(Smart) abc a
<element name = "title" type = "Title.Type" />

⇒ Datatypes:

SIMPLE TYPE	COMPLEX TYPE
• No sub-elements	• Sub-elements ofde
• No attributes of ele.	• Attr. possible.

⇒ Pre-defined datatypes are available, like- string, token, byte.

⇒ <element name = "Title" type = "string" />

(default namespace available) ← no need to use xsd: because it is primitive datatype

⇒ Derived simple type.

<xsd: simpleType name = "Roll No Type">
<xsd: restriction base = "xsd: integer">
Declaration
<xsd: minInclusive value = "1" />
<xsd: maxInclusive value = "130" />
</xsd: restriction>
</xsd: simple Type>

⇒ <xsd:pattern value="\d{2}[A-Z]{3}"/>
2 digits followed by 3 alphabets.

⇒ ComplexType
<xsd:complexType name="Student Type">
<xsd:sequence>
<xsd:element name="name" type="xsd:String"/>
</xsd:sequence>
<xsd:attribute name="category" type="xsd:String"/>
</xsd:complexType>

(simpleType) → can reference both simple & complex types
element of type StudentType can have multiple sub-elements
because attribute can't have sub-elements → can reference only simple type
parameter the seq. of the tag while using it

use: <Student>
<name> abc </name>
<RollNo> 1 </RollNo>
<Semester> 5 </Semester>
<email> abc.xyz@gmail.com </email>
</Student>

Occurance:
<element name="email" type="String"
minOccurs="1" maxOccurs="6"/>

Attribute Enumeration
<xsd: attribute name="category" default="BE"/>
<xsd: simpleType base="xsd:String">
<xsd: enumeration value="BE"/>
"ME"/>
</xsd: simpleType>
</xsd: attribute>

List Type

```
<xsd:simpleType name="IDSList">
  <xsd:list itemType="IDType"/>   → defined earlier (pattern*)
</xsd:simpleType>
```

→ `<IDSList> 09ABC 10PQR </IDSList>`

Q. Define student where stdID, stdName (FN, MN, LN), stdMarks, stdMobNo. (1 or more. max - 3).
Create ID element
Then create XML instance doc.

→

```
<xsd:complexType name="StudentType">
  <xsd:sequence>
    <xsd:element name="stdID" type="IDType"/>
    <xsd:element name="name" type="NameType"/>
    <xsd:element name="marks" type="xsd:integer"/>
    <xsd:element name="mobNo" type="xsd:integer"
                 minOccurs="1" maxOccurs="3"/>   ← can use pattern.
  </xsd:sequence>
</xsd:complexType>
```

* (write before defining StudentType)

```
<xsd:complexType name="IDType">
  <xsd:restriction base="xsd:String">
    <xsd:pattern value="\d{2}[A-Z]{2}\d{3}"/>
  </xsd:restriction>
</xsd:complexType>

<xsd:complexType name="NameType">
  <xsd:sequence>
    <xsd:element name="fname" type="xsd:String"/>
```

<xsd:element name="mname" type="xsd:string"/>
<xsd:element name="lname" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>

16/7/19 <xsd:element name="std" type="StudentType">
</xsd:element>

UnionType : Enables an element to be one or more instances of one type drawn from the union of multiple atomic and list types

<xsd:simpleType name="cityType">
<xsd:union itemType="cityPostCity"/>
</xsd:simpleType>

⇒ Explicit type - name is given to a type defined earlier

<!-- define -->
<xsd:complexType name="StudentType">

</xsd:complexType>

<!-- declare -->
<element name="std" type="StudentType">
</element>

can be used by any element throughout the xsd doc.

Implicit type: defination & declaration together

```
<element name = "Std">
  <xsd: complexType name = "StudentType">
    :
  </xsd: complexType>
</element>.
```

⇒ xsd:All gives the liberty that all the elements can occur in any order.

group

```
<xsd: group name = "shipAndBill">
  <xsd: sequence>
    <xsd: element name = "shipTo" type = "addr"/>
    <xsd: element name = "billTo" type = "addr"/>
  </xsd: sequence>
</xsd: group>
```

choice

```
<xsd: choice>
  <xsd: group ref = "shipAndBill" />
  <xsd: element name = "singleAddr"
                type = "addr" />
</xsd: choice>.
```

Choosing any one of two, i.e. either enter 2 addr. (ship & bill) or specify one addr. for both.

17/7/19 Restriction on simple type (Derived simple type)

```
<simpleType name="SmeText">
  <restriction base="string">
    <length value="140" fixed="true"/>
  </restriction>
</simpleType>
```

Pattern: (a|b)+x : ax, bx, aax, abx, bax

for email

```
<pattern value = "*@*.(com|org|net)>
```

⇒

BPEL
Web Service
WSDL, SOAP
XML

Key terms of SOA

Service - repeatable task (business task)
Ex - check customer credit

Service orientation - A way to integrate your business as linked services with the unique outcome

SOA - Architectural style that supports service orientation

Composite app^n: Integrating two services from different domains separately built on SOA for your own business goals

⇒ Service oriented (platform & prog. lang. independent)

Service oriented		
Reusability	Tech. independence	Loose coupling

Object oriented

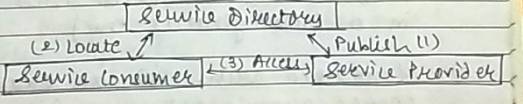
⇒ n services : API's needed = $n(n-1)$
Thus, we need a standard to reduce this complexity.

SOA - An approach for building distributed systems that deliver application functionality as services to either end user appln or other services.

⇒ XML is interpreted by every machine (Biggest advantage of XML)
[For SOA, it is its Reusability]

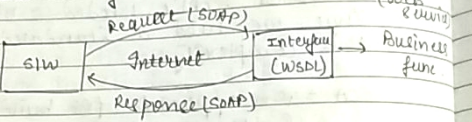
SOA define:
i) Software assets ≈ Services
ii) Provides a standard way of representing & ~~integrating~~ interacting with software assets.
iii) Individual modules are the building blocks used to develop other appln.

Elements of SOA

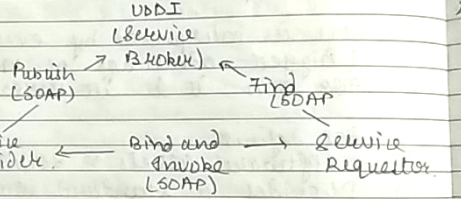


Characteristics of Web Service:

→ identified by a URL
→ whose public interfaces & bindings are defined
→ described using XML.



UDDI → registry



⇒ When we have a homogeneous env (all modules built using same prog. lang. or on same platform) then interaction between these modules is easy and thus, SOA is not used. (modules are converted into services in a heterogeneous env.)

⇒ Monolithic arch. is an example of coarse grain arch. as its resource

utilization is more the modules not being used are also loaded into the ~~mem~~ OS.
Microkernel is a fine grain arch where the modules which are req. are only loaded with OS, thus enhancing resource utilization.

⇒ Heterogeneous system's modules are wrapped (adding an overhead) which can be used by the defined standard for communication with other modules.

WSDL is like an interface, having declaration of methods and also contains the location.

Steps:
i) Service provider publishes WSDL and registers with UDDI
ii) Service Requestor gets the WSDL of a particular service.
iii) Service Requestor ~~needs to find~~ then invokes & accesses the service from the provider.

Advantages of SOA:
- Standards-based
- Cross platform and cross-language
- Widely supported
- Message-oriented (+ext based: XML)
(JRMP - binary based - heavier than text ba

WSDL (Web Service Description Language)
- Open standard for describing interfaces to services.

Characteristics:
i) Describes data expected to be sent and received
ii) What the service can do
iii) How to access it (URI/URL)

SOAP (Simple Obj. Access Protocol)
- It an application layer protocol which is inheritantly connection oriented because its underlying protocol is TCP

Skeleton of SOAP: Envelop — Header (optional), Body

SOAP SOA provides business agility in 3 ways:
- Loosely coupled
  (Services do not require the same technological implementation at each end of connection).
- Reuse
  Return of Investment (ROI) for SOA is quite high due to reuse of S/W, H/W, procedures etc.
- Extensibility
  It is defined as the ability to easily

expand internal operations with new functionalities and to easily access organizations outside the enterprise.

29/7/19 Key principles of SOA:

i) Loose coupling
ii) Service Contract (WSDL doc. containing the methods for impl)
    - Code first approach
    - Contract first app
iii) Autonomy
iv) Abstraction. (Service hide logic from outside world)
v) Reusability
vi) Composability
vii) Statelessness
viii) Discoverability

Creating web service.

Project → Web App → Web Service

Design view → Add operation
(Specify parameters)