# Programmazione Web

# Lez. 12

# ReST, XML Avanzato, Web Services

**Giuseppe Psaila**

*Università di Bergamo*

*giuseppe.psaila@unibg.it*

# ReST

# ReST

- Representational
- State
- Transfer

# Che Cosa È?

- È  uno stile architetturale
- Un modo di organizzare l’architettura con cui i servizi sono definiti
- Purtroppo, viene confuso con i Web Service richiamabili con il protocollo HTTP

# Chiariamo

- Un Web Service è un servizio orientato ad altre applicazioni
- Che espone delle API (Application Programming Interface) usate da altre applicazioni per ottenere servizi, fornendo e/o ottenendo dati
- Siccome viene richiamato effettuando richieste HTTP, prende il nome di «Web Service»

# Servizi Web e ReST

- Un servizio web può essere sviluppato con lo stile architetturale ReST
- Ma può essere sviluppato senza seguire questo stile architetturale

# ReST e ReSTful

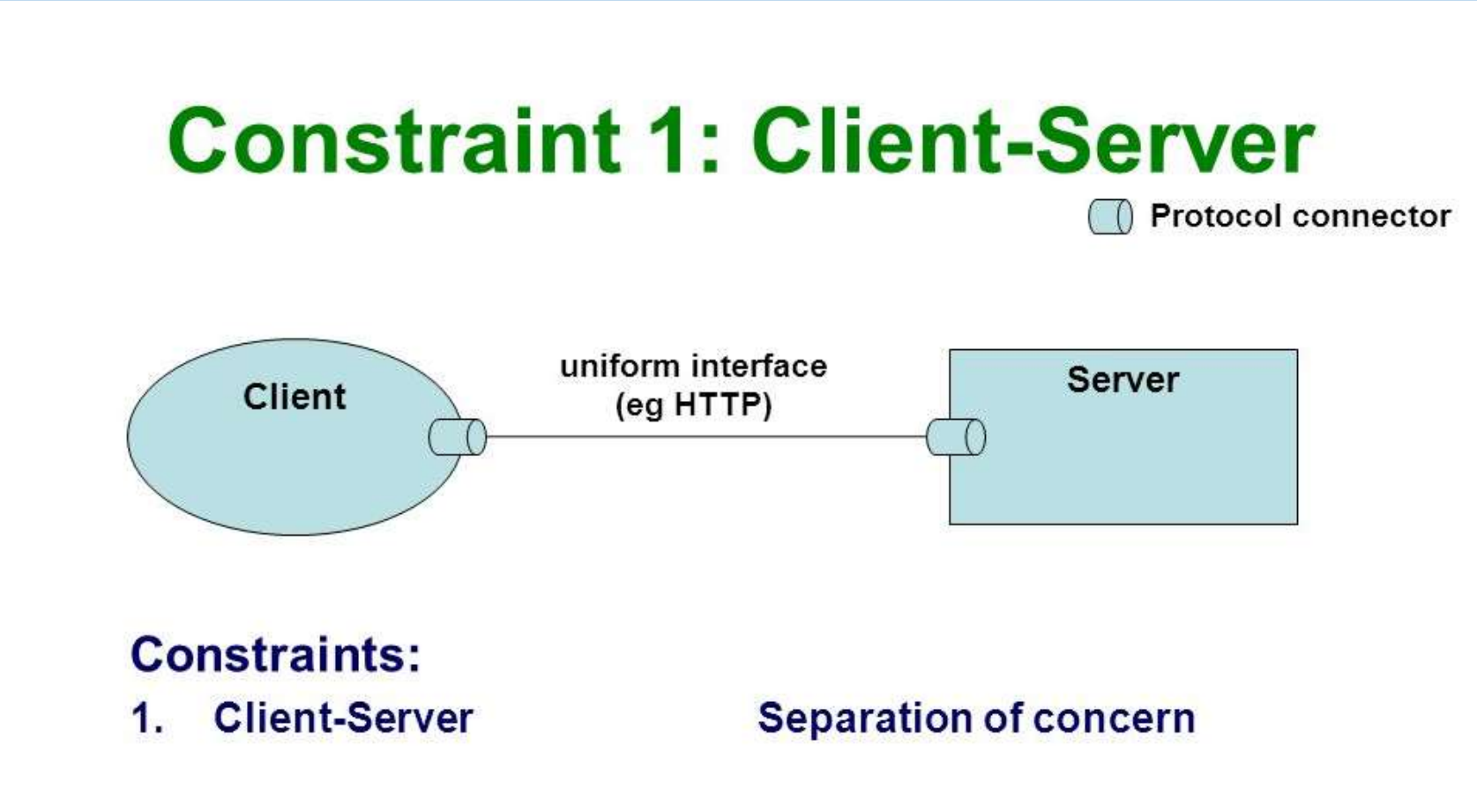
- Lo stile ReST impone alcuni vincoli sul modo in cui realizzare un architettura basata sui servizi
- Un servizio «ReSTful» o un’architettura ReSTful indicano che il servizio o l’architettura rispettano i vincoli/principi definiti dallo stile ReST
- I vincoli/principi di ReST sono focalizzati sul modo in cui i dati vengono trasmessi

# Principi ReST

- **Client-Server**
  Un architettura ReST è client-server. Cioè un servizio fa da server ad un altro servizio/applicazione che funge da client
- Questo per separare chiaramente gli ambiti di intervento del client e del server

# Principi ReST

- Client-Server



# Principi ReST

- **State-less**
  La comunicazione deve essere senza stato, cioè non deve essere basata sullo scambio di messaggi multipli tra client e server
- Lo scambio di messaggi multipli, anche distanti nel tempo, costringe il server a mantenere lo stato della comunicazione con molti client
- State-less: il lavoro richiesto al server si esaurisce con la risposta alla richiesta

# Principi ReST

- **Cache**
  Visto che un servizio fornisce dei dati, se le condizioni che portano a fornire quei dati non sono cambiate, due richieste vicine nel tempo che riguardano la stessa risorsa dovrebbero ottenere esattamente la stessa risposta
- Questo consente di sfruttare la memoria cache dei browser o dei proxxy per non rieseguire le richieste (usando la vecchia risposta alla stessa richiesta)

# Principi ReST

- **Interfaccia Uniforme**
  I contenuti dei messaggi scambiati tra client e server devono essere uniformi
- Si intende che devono essere basati su un formato standard
- Indipendente, il piùù possibile, dall’applicazione

# Principi ReST

- **Interfaccia Uniforme: Risorse**
  Una risorsa è un oggetto o la rappresentazione di qualcosa di significativo nel dominio applicativo. È possibile interagire con le risorse attraverso le API.
- Una richiesta al servizio richiede una risorsa
- Esempio: un prodotto di Amazon

# Principi ReST

- **Interfaccia Uniforme: Manipolazione attraverso Rappresentazioni**
  La stessa risorsa può essere rappresentata inmolti modi: XML, JSON, PNG
  Il servizio può fornire rappresentazioni diverse per la stessa risorsa
- Esempio: il prodotto Amazon è rappresentato con un XML o con un JSON
  Il client usa quella rappresentazione per gestire il prodotto

# Principi ReST

- **Interfaccia Uniforme: Hypermedia come motore dell'applicazione**
  Le azioni sulle risorse sono guidate da link, presenti nella rappresentazione delle risorse stesse
- Esempio: nella rappresentazione del prodotto,
  - Un link rappresenta l'azione per avere maggiori dettagli
  - un link rappresenta l'azione per acquistarlo

# Esempio: XML

```
<album>
 <title>the title</title>
 <code>1234</code>
 <description>A new piece of art</description>
 <link rel="/artist" href="/artist/blackMen"/>
 <link rel="/purchase" href="/purchase/1234"/>
</album>
```

# Gli Elementi link

- Il documento XML descrive una risorsa di tipo «album»
- I due elementi «link» descrivono due lik associati con la risorsa, cioè due azioni:
  - Il primo consente di ottenere la descrizione dell'artista
  - Il secondo consente di effettuare l'acquisto dell'album
- Si noti l'attributo «rel», che indica per quale motivo il link è associato al documento, cioè l'azione possibile

# Ecco il Significato di ReST

- Ecco perché ReST, cioè Representational State Transfer
- Perché il server trasferisce al client la rappresentazione dello stato della risorsa, con associati i link che descrivono tutte le azioni possibili che si possono effettuare sulla risorsa stessa
- L’uniformità è data dal fatto che si usano gli URI

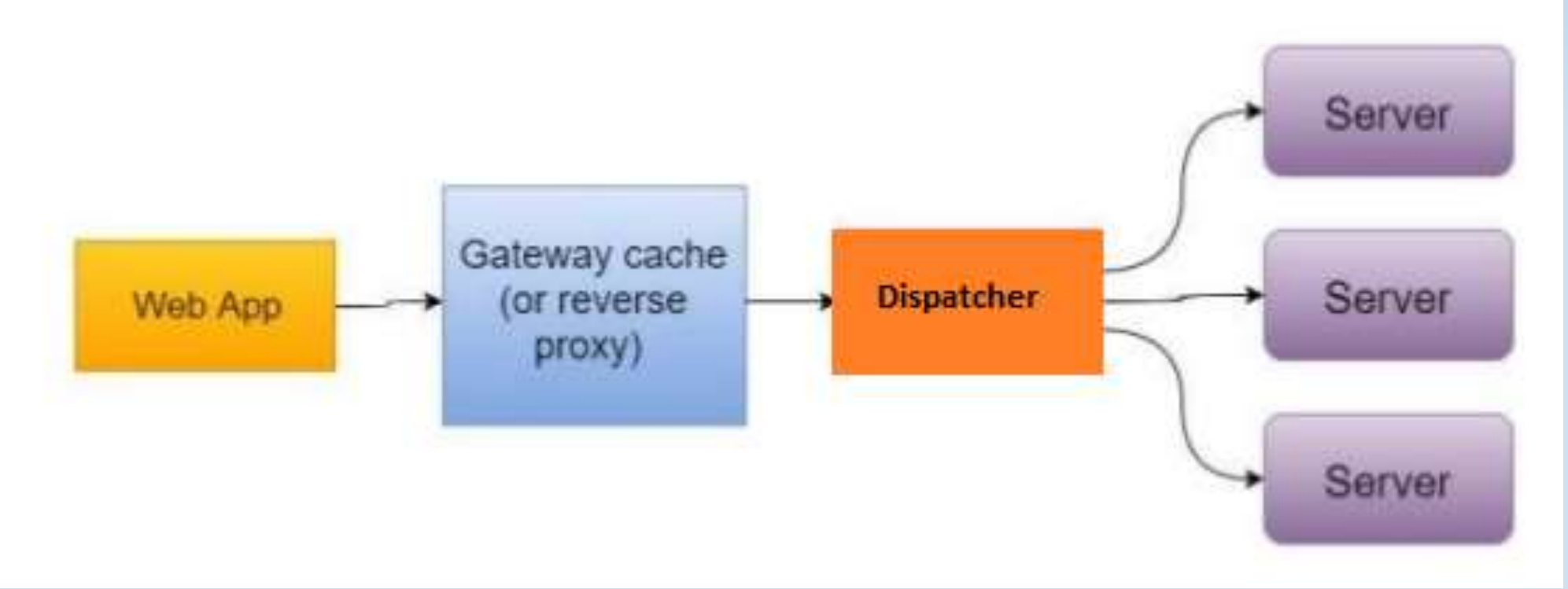
# ReST e Micro-Services

- L’approccio ReST è l’ideale nei sistemi basati su micro-services
- Perché un micro-servizio fornisce la rappresentazione di una risorsa con le azioni possibili su di essa e i link ai micro-services che le eseguono

# ReST e Micro-Services

- Un micro-servizio è associato ad un tipo di risorsa che gestisce, sarà il **dispatcher** delle richieste web a indirizzare la richiesta verso il servizio e il server appropriati
- In questo modo, si ottiene un elevatissimo grado di componibilità dei servizi

# ReST e Micro-Services



# XML o JSON?

- XML fornisce di suo il concetto di «link»
- Nel senso che è universalmente accettato un elemento «link» con le caratteristiche viste prima
- E JSON?
- Si può fare riferimento alla proposta *Hypertext Application Language (HAL)*

# Esempio: JSON

```
{ "type": "album",
   "title": "the title",
  "code": "1234",
  "description": "A new piece of art",
  "_links": { "artist": "/artist/blackMen"/",
                    "purchase": "/purchase/1234" }
}
```

# Esempio: JSON

- Il campo "_links" contiene tutti i link associati all’album, dove il nome del campo descrive il tipo di azione possibile

# Riferimenti

- Introduzione a ReST
  **`https://italiancoders.it/introduzione-a-rest/`**

# Namespace n XML

# Namespace: Spazio dei Nomi

- Un «Namespace» raccoglie una serie di nomi o simboli
- Il concetto viene usato in molti ambiti, per esempio nel C++
- In XML, un namespace definisce elementi specifici
- Per poter usare questi elementi, occorre indicare a quale namespace appartengono

# Prefisso del Namespace

- Ogni namespace usato nel documento ha un prefisso
- Il prefisso deve precedere il nome dell'elemento
- Prefisso e nome sono separati da «:»
- Esempio (da specifica SOAP)

**`soap:Envelope`**

prefisso:nome elemento

# Prefisso e Namespace

- Ma i prefissi vanno definiti
- La prima volta che un prefisso viene usato, occorre dire a quale namespace appartiene
- Con uno strano attributo:
  **`xmlns:`***prefisso*
- Esempio:

```
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
```

# xmlns:*prefisso* e suo valore

- Questo attributo dice che il *prefisso* verrà usato da lì in avanti
- Ma che valore ha?
- Il valore è l'URI dello standard
- URI: Uniform Resource Identifier
  Si tratta dell'evoluzione del concetto di URL (Uniform Resource Locator) per **identificare** una risorsa, non per trovarla

# URI

- Ma un URI ha la stessa forma di un URL
- L'idea è questa:
  - Il processore analizza il documento
  - Vede l'URI associato a **`xmlns:`**
  - Se lo conosce, è in grado di processare gli elementi appartenenti a quel namespace
  - Se non li conosce, li scarta/ignora

# Perché i Namespace?

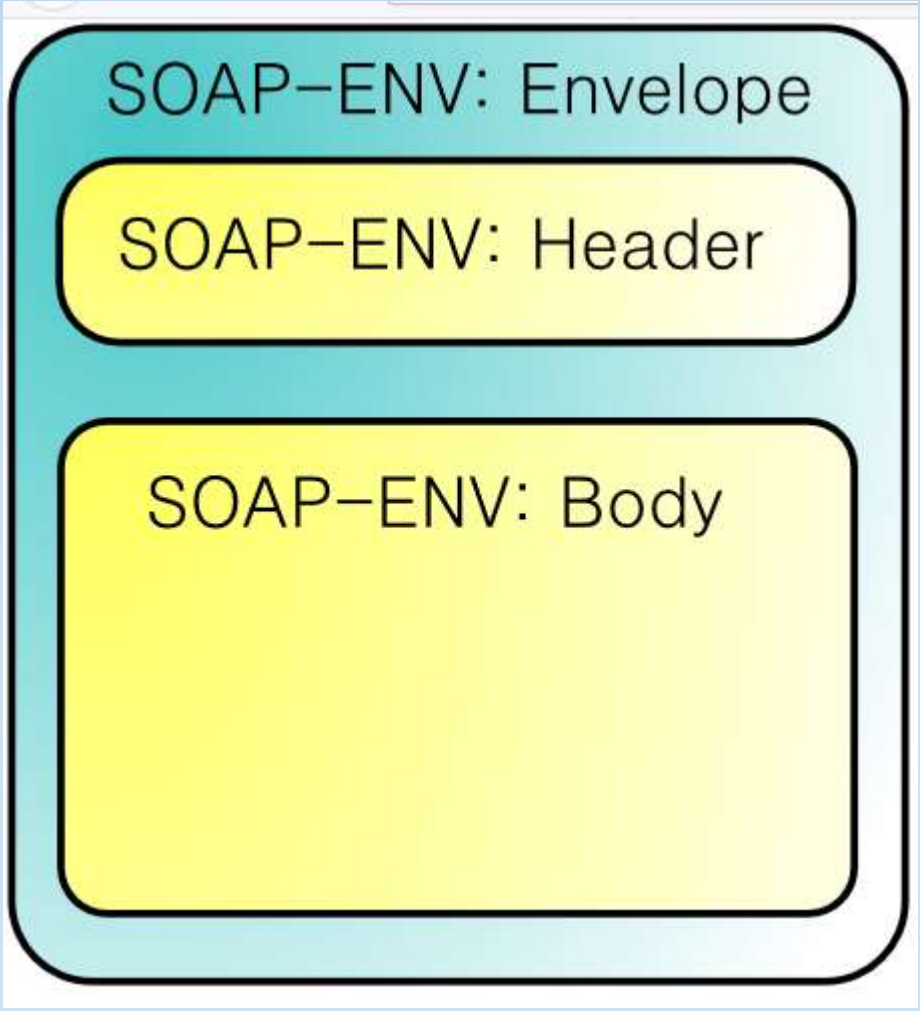
- Perché così si possono integrare nello stesso documento elementi appartenenti a namespace diversi
- Cosa impossibile da fare con il DTD
- Inoltre, si può dichiarare esplicitamente a quale definizione del documento si fa riferimento

# Esempio: SOAP

- Simple
  Object
  Application
  Protocol
- È un protocollo del W3C usto per lo scamabio di messaggi tra sistemi informativi
- I messaggi sono documenti XML

# Messaggi SOAP

- Envelope
  che contiene
  - Header
  - Body



# SOAP BODY

- Nel corpo del messaggio viaggia il contenuto effettivo della comunicazione
- Ma il formato da inviare non è relativo al protocollo SOAP, che è generico
- Un po’ come la busta della posta tradizionale: è neutra rispetto al contenuto

# Esempio SOAP

```
<soap:Envelope xmlns:soap="http://...">
 <soap:Body>
   <getProductDetailsResponse xmlns=
      "http://magazzino.example.com/ws">
…
```

# Esempio SOAP

```
…
   <getProductDetailsResponse xmlns=
    "http://magazzino.example.com/ws">
    <getProductDetailsResult>
     <productName>Matita</productName> …
    </getProductDetailsResult>
   </getProductDetailsResponse>
  </soap:Body>
</soap:Envelope>
```

# XmlNs Senza Prefisso

- Definisce il namespace per gli elementi senza prefisso
- Serve per dire a quale specifica/standard fanno riferimento

# Ricapitoliamo

- Il Server del destinatario riceve un messaggio SOAP
- L’handler del protocollo SOAP riceve il contenuto XML del messaggio
- Sa gestire i suoi elementi, non gli altri
- Estrae il frammento nel body e lo passa al componente software del sistema informativo, che è in grado di processarlo

# XML Schema

# XML Schema (o XSD)

- XML Schema è nato per sostituire il DTD
- Ma non solo lo sostituisce, fornisce concetti che DTD non fornisce
  - Tipi di dati per gli attributi
  - Riuso di strutture
- Nel seguito vedremo solo alcuni aspetti di XML Schema, perché è troppo vasto

# StrutturaBase

- Il prefisso del namespace è xs

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
 xmlns:xs="http://www.w3.org/2001/XMLSchema">
...
</xs:schema>
```

# Definire un Elemento

- Se l'elemento da definire ha un contenuto solo testuale, la sua definizione è molto semplice

```
<xs:element name="orderperson"
    type="xs:string"/>
```

- Con type si può definire il tipo del contenuto, in questo caso è una stringa

# Tipi di Dato

Vari tipi di dato sono disponibili (alcuni)

- xs:positiveInteger
- xs:integer
- xs:decimal
- xs;string
- xs:date
- xs:time
- xs:dateTime

# Definizione di Elemento Semplice

- Esempio:

```
<xs:element name="title"
      type="xs:string" />
```

Elemento che contiene solo testo. Esempio:

```
<title>My Title</title>
```

# Definizione di Elemento Complesso

- Si definisce un tipo complesso

```
<xs:element name="shiporder">
  <xs:complexType>
        ...
  </xs:complexType>
</xs:element>
```

# Definizione di Elemento Complesso

- Che cosa c’è im un tipo complesso?
- <xs:sequence>
  definisce sequenze di elementi
- <xs:all>
  indica gli elementi che devono essere presenti, indipendentemente dall’ordine

# Definizione di Elemento Complesso

- <xs:choice>
  Uno solo degli elementi indicati può occorrere nel contenuto
  Può essere usato anche dentro <xs:sequence>

# Definizione di Elemento Complesso

- Per vincolare la molteplicità, sia xs:element che gli altri elementi prevedono due attributi opzionali
- minOccurs
  valori "0" o "1"
- maxOccurs
  valore non negativo o "unbound"

# Riuso di Elementi

- È possibile definire gli elementi per poi essere usati più volte
- Come usarli?
- <xs:element ref="nome"/>

# Definizione di Attributi

- È possibile definire gli attributi degli elementi
- <xs:attribute name=… type=… use=…/>
- use è opzionale e vale:
  - "optional"
  - "required"

# Riuso di Attributi

- È possibile riusare attributi generici
- <xs:attribute ref="nome"/>

# Altri Costrutti

- Si possono definire anche dei tipi riusabili
  - Per il contenuto degli elementi
  - Per gli attributi
- Si possono definire classi di tipi
- Si possono definire gerarchie
- Si possono definire elementi partendo da classi di tipi

# Correttezza e Validazione

- Una specifica XSD sostituisce e potenzia il DTD
- Quindi, un documento viene detto «Valido» rispetto alla specifica XSD
- Come validare? Con i parser validanti XML Schema
  XML-Schema Validating Parser

# Esempio

- Vedi file XML_Schema.pdf

# WSDL

# WSDL

- Web
- Service
- Definition
- Language

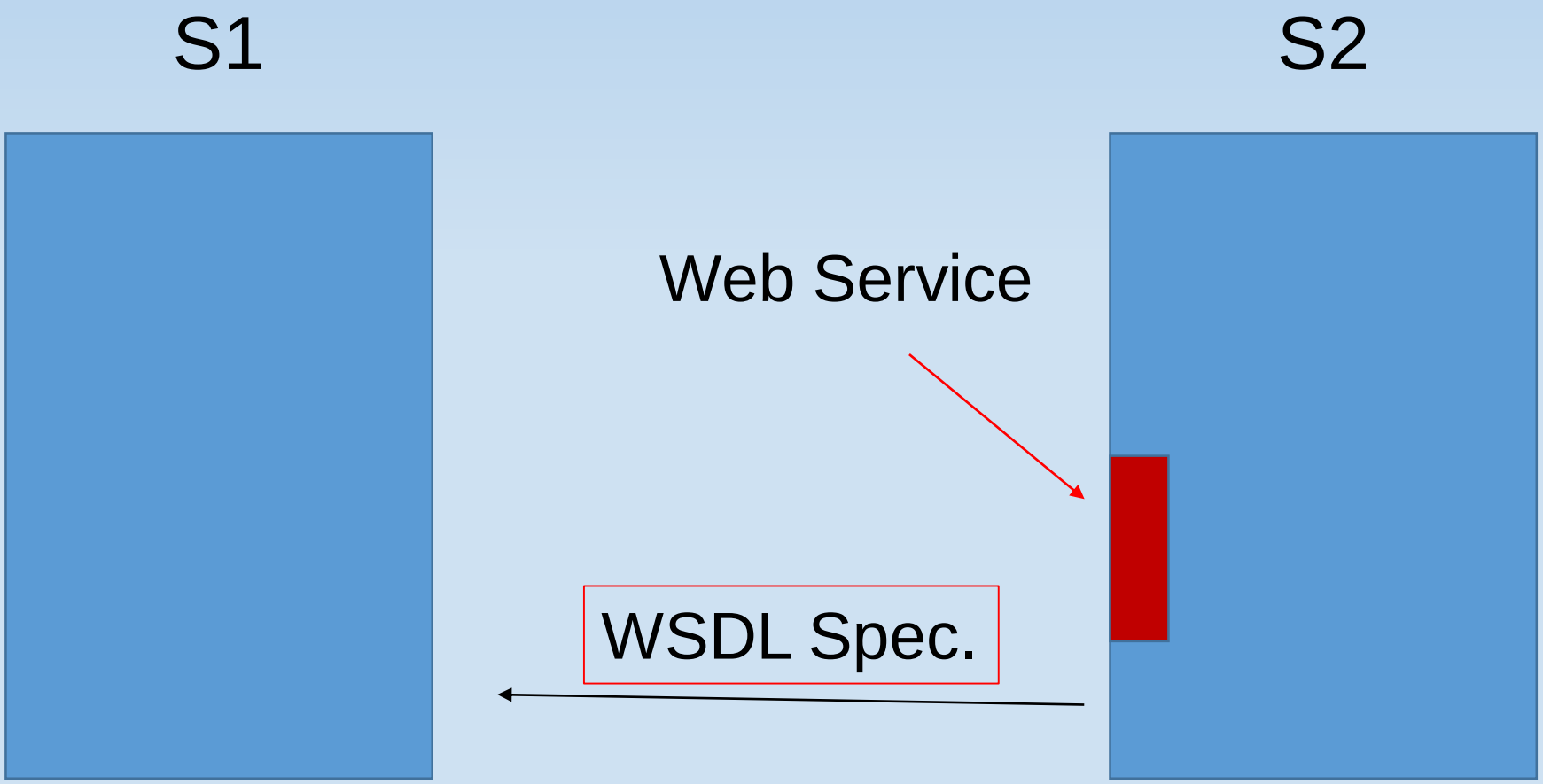
# Scopo

- Definire in modo formale un web service
- Per facilitare lo scambio dei dati con il web service
- Conseguenza: sviluppo di librerie e strumenti per effettuare la comunicazione in modo automatico, con poche linee di codice

# Visione

- Supponiamo di avere due sistemi informativi S1 e S2
- S2 deve ricevere dei dati da S1, da incorporare nelle sue strutture dati
- S2 espone un web service e genera una specifica WSDL che invia a S1

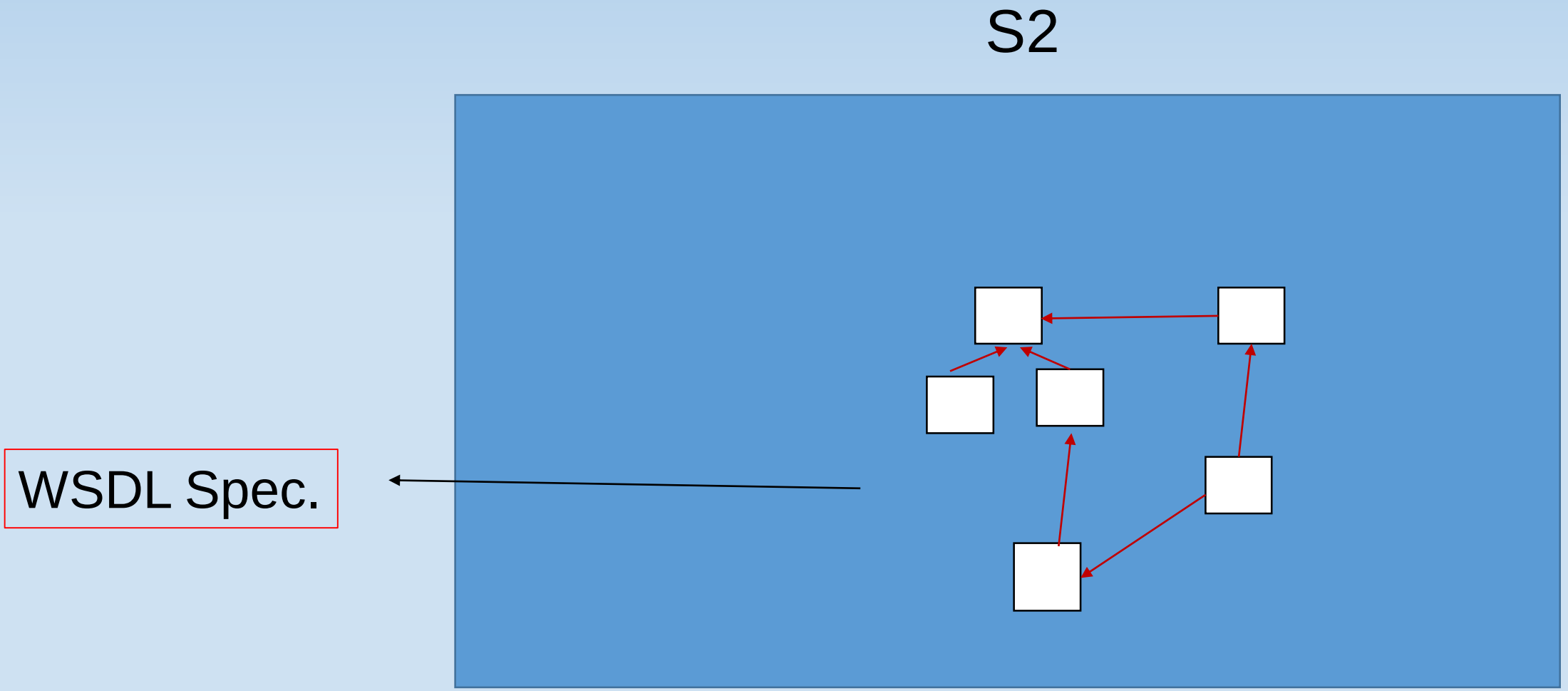
# Visione



# Visione

- Ma chi scrive la specifica WSDL?
- Nessuno,
- Viene generata automaticamente dalle strutture dati object-oriented che devono essere popolate con i dati ricevuti dal web service
- Partendo o dalle definizioni delle classi o dalle definizioni del database

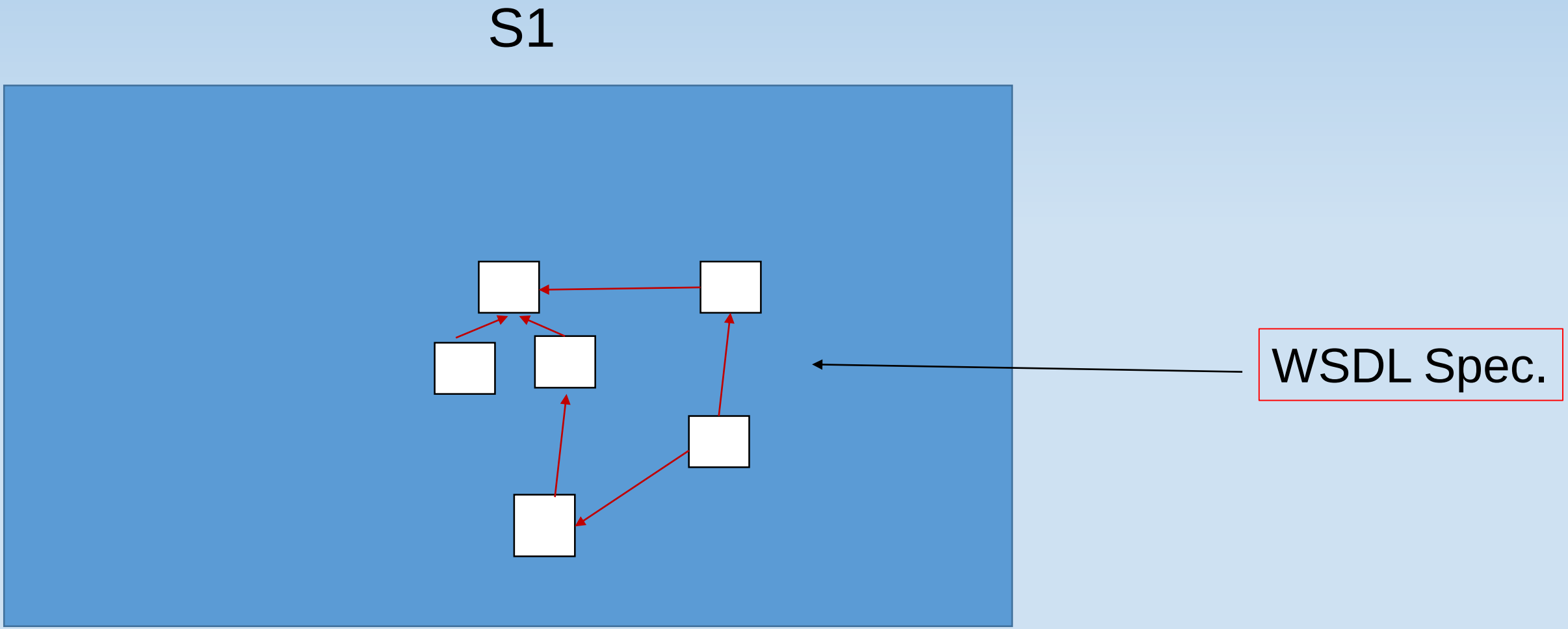
# Visione



# Visione

- S1 riceve la specifica WSDL
- Da essa si genera la definizione delle strutture dati che devono contenere i dati da inviare
- Nel linguaggio usato per implementare S1

# Visione

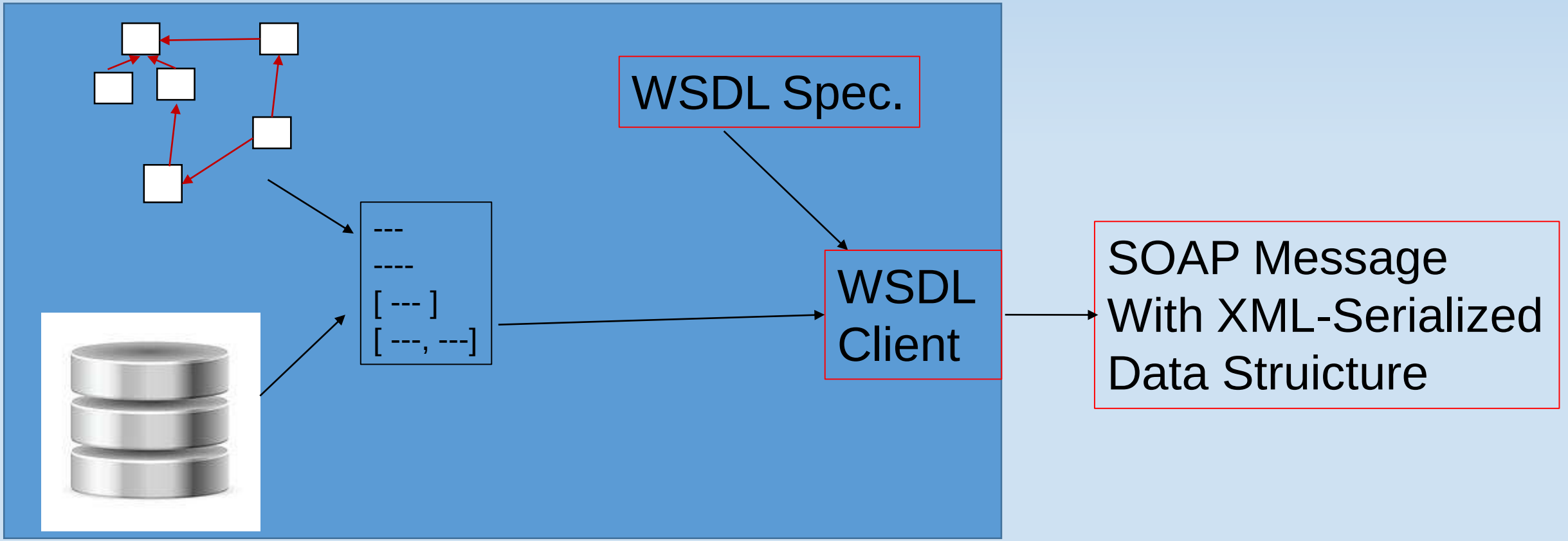


# Visione

- I programmatori di S1 popolano la struttura dati con i dati da inviare
- Usando le definizioni ottenute dalla specifica WSDL
- Il client WSDL in S1 serializza la struttura dati, usando la specifica WSDL per generare il contenuto XML del messaggio

# Visione

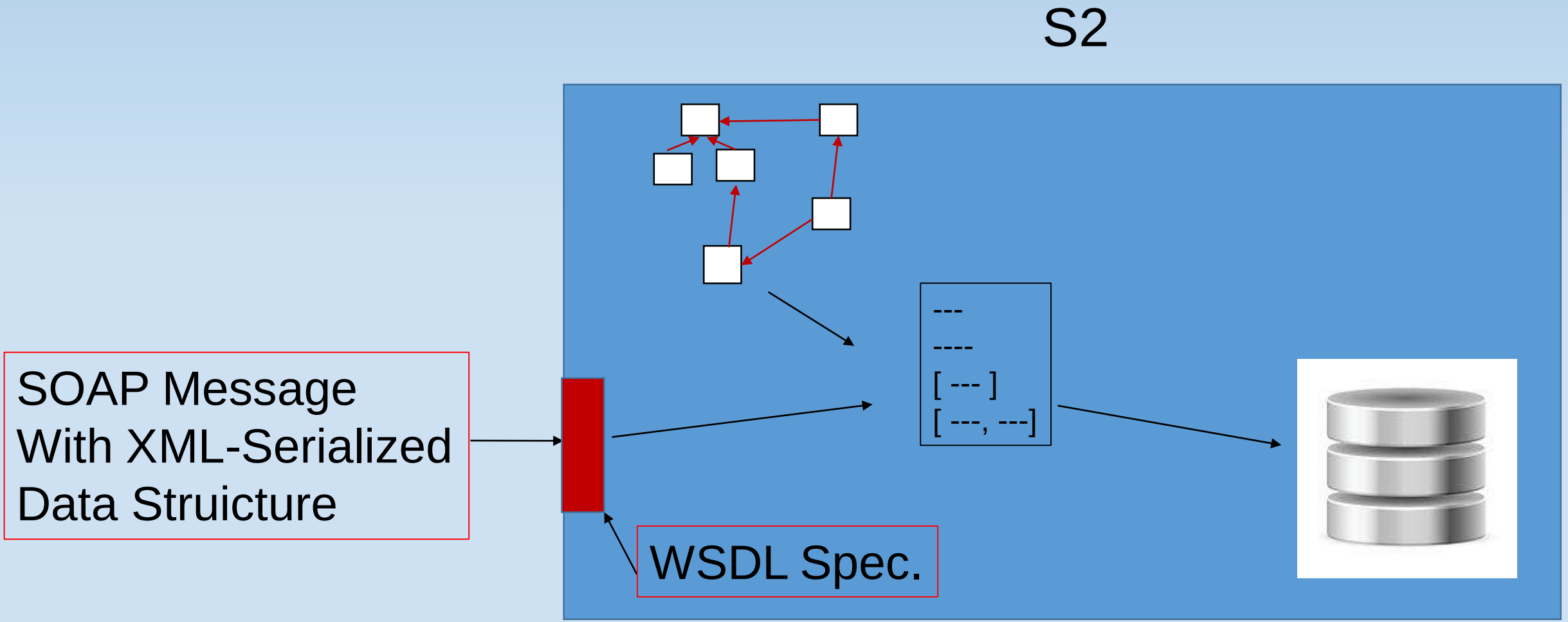
S1


WSDL Spec.
---
----
[ --- ]
[ ---, ---]
WSDL
Client
SOAP Message
With XML-Serialized
Data Struicture

# Visione

- Il web service esposto da S2 riceve il messaggio
- Usando la specifica WSDL deserializza il messaggio
- Popola la struttura dati interna a S2, nel suo linguaggio di programmazione
- Questa struttura dati verrà utilizzata dal resto del codice di S2

# Visione


S2
SOAP Message
With XML-Serialized
Data Struicture
---
----
[ --- ]
[ ---, ---]
WSDL Spec.

# Dettaglio di WSDL

- Non lo vediamo, perché allo stato attuale è inutile conoscerlo. Se siete interessati trovate facilmente molti documenti
- Che cosa occorre sapere?
- Lo vediamo nelle prossime slide

# XML

- La specifica WSDL è basata su XML
- Fa parte di un namespace specifico
- Definisce i contenuti dei messaggi e come comunicare con il web service esposto

# XSD

- Il contenuto dei messaggi è definito tramite una specifica XSD
- Quindi, WSDL si appoggia allo standard XML Schema e usa il suo sistema dei tipi
- Quindi, adotta una visione a oggetti

# HTTP

- Il protocollo usato per inviare i messagi è HTTP
- Trattandosi di web services, non poteva essere diversamente
- Il lato server del web service si appoggia ad un HTTP Server, occorre quindi aprire una porta specifica

# SOAP

- Il contenuto dei messaggi (inviati via HTTP) è basato sul protocollo SOAP
- La envelope di SOAAP contiene la serialzzazione delle strutture dati
- La serializzazione è fatta con XML

# Vantaggi

- Tutti i passaggi intermedi sono gestiti in automatico
- I programmatori lavorano direttamente sulle strutture dati di propria competenza
- Senza conoscere minimamente il WSDL
- La specifica WSDL viene generata automaticamente