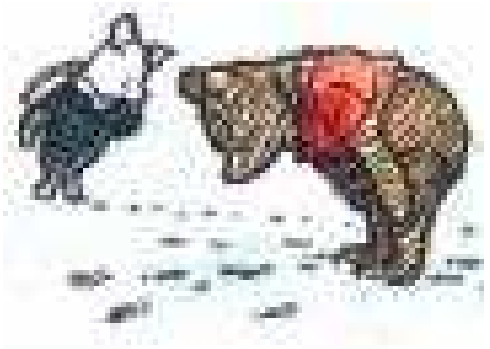
WSDL 2.0

In which we go hunting and nearly catch a *WSDL* (with apologies to A.A. Milne)

Paul Brebner
CSIRO ICT Centre
Canberra
Paul.Brebner@csiro.au

Tracking WSDL, or WSDL for “tracking” services



"The tracks!" said Pooh.

"A third animal has joined the other two!“

"Pooh!" cried Piglet "Do you think it is another Woozle?“

"No," said Pooh, "because it makes different marks.

It is either Two Woozles and one, as it might be, Wizzle, or Two, as it might be, Wizzles and one, if so it is, Woozle.…”

Overview

- Introduction: WSDL History, Interfaces
- WSDL Model
  - Abstract
    - Interface
    - Operations
    - MEPs
    - Messages
  - Document
  - Concrete
    - Binding
    - Service
    - Endpoint
- Extras

Tracking WSDL (history)

- 2000: WSDL 1.0
- 2001: WSDL 1.1
  - W3C Note
- 2002-2006: WSDL 1.2 -> 2.0
  - Candidate recommendation
  - Implementations and Interoperability
  - Proposed recommendation
  - Recommendation
  - Could take a while still ☺

WSDL 2.0

The Web Services Description Working Group was chartered to describe:

- the messages
- the message exchange patterns
- the protocol binding

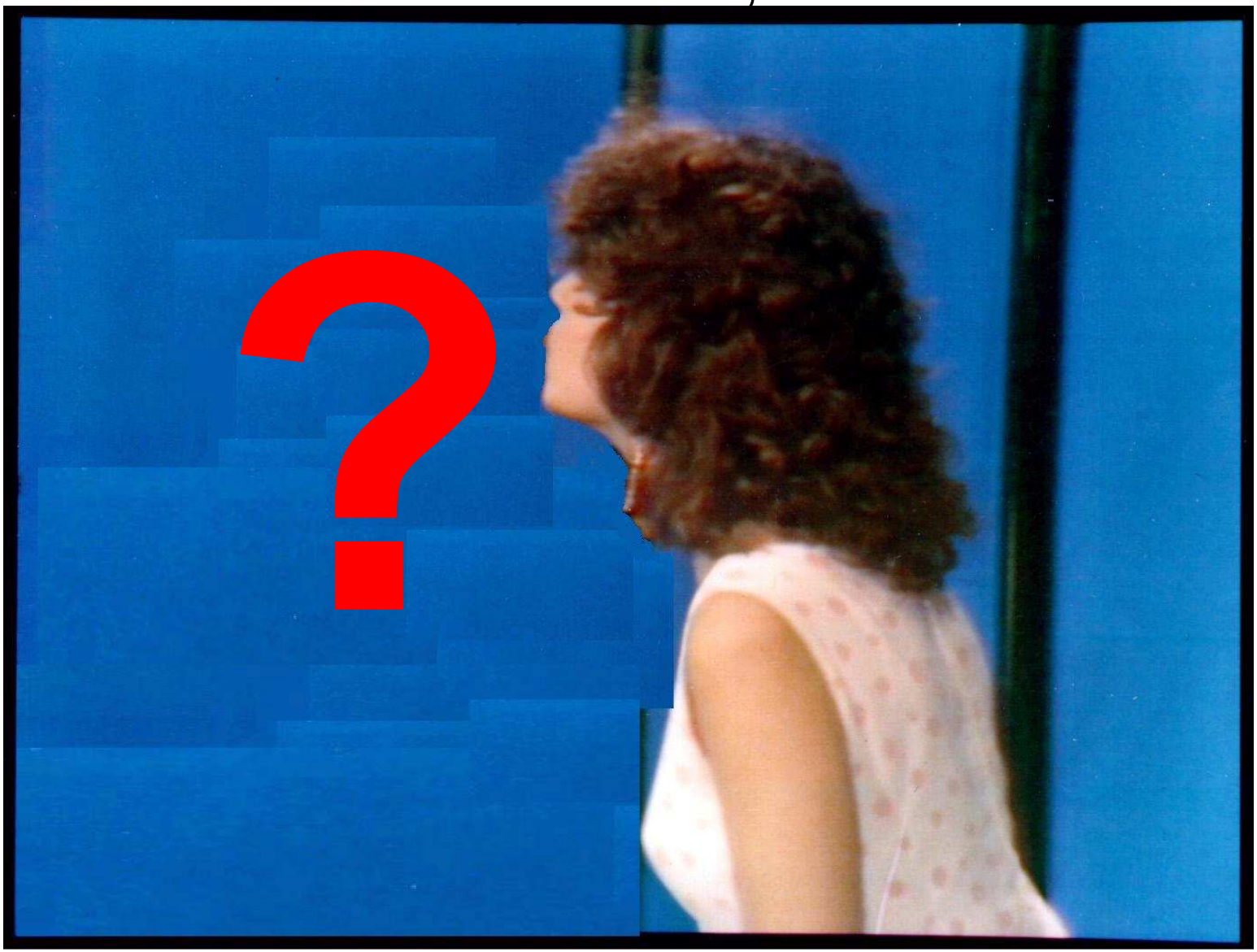
for the **interface…**

the boundary across which web services and their clients communicate

Charter for the WS Description Working Group, 2002.

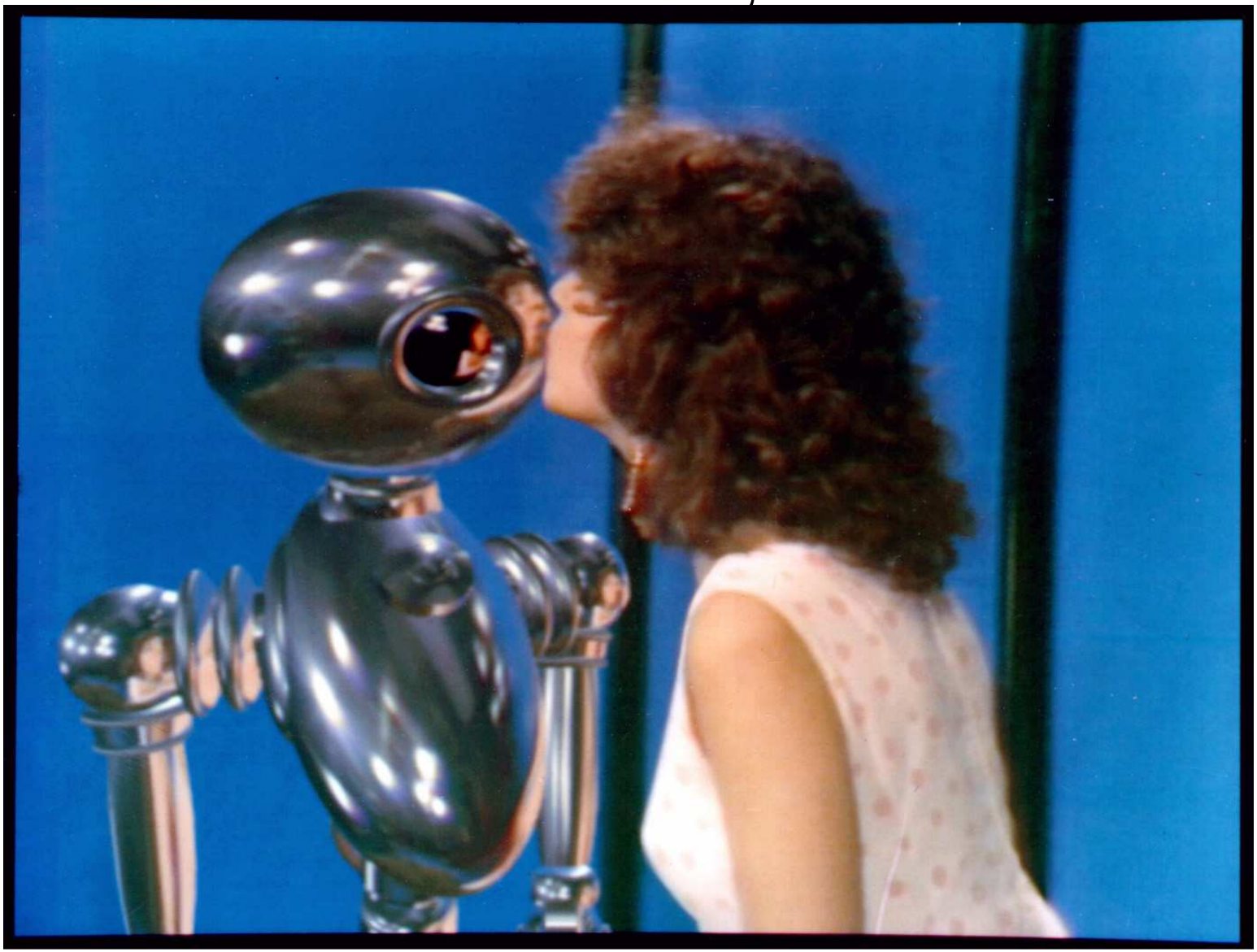
“Interface”

Lance Williams, 1985



“Interface”

Lance Williams, 1985



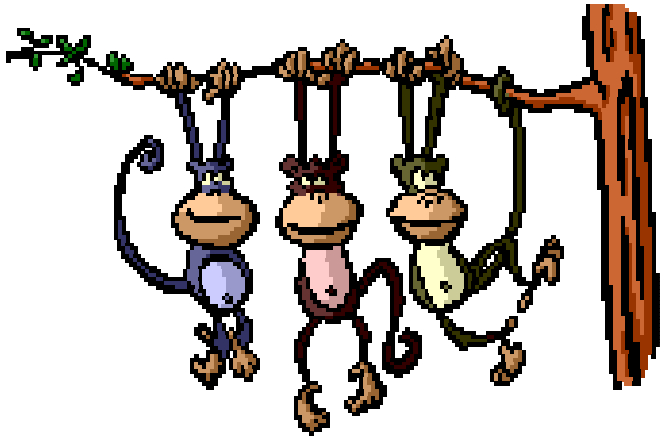
Clients

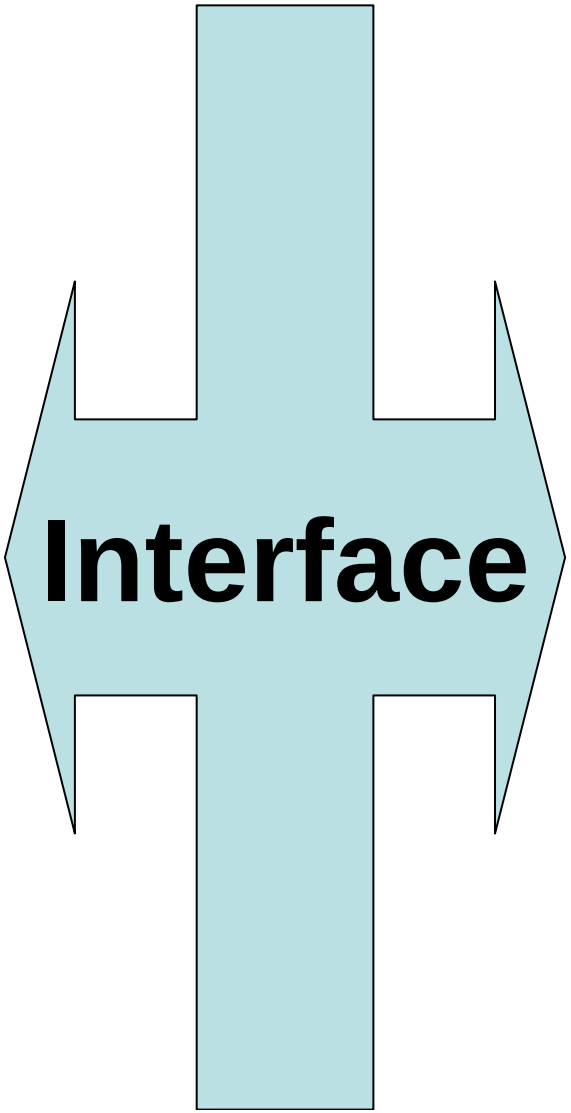
Interface

Implementation

Isolates clients …

from implementation





Clients

Interface

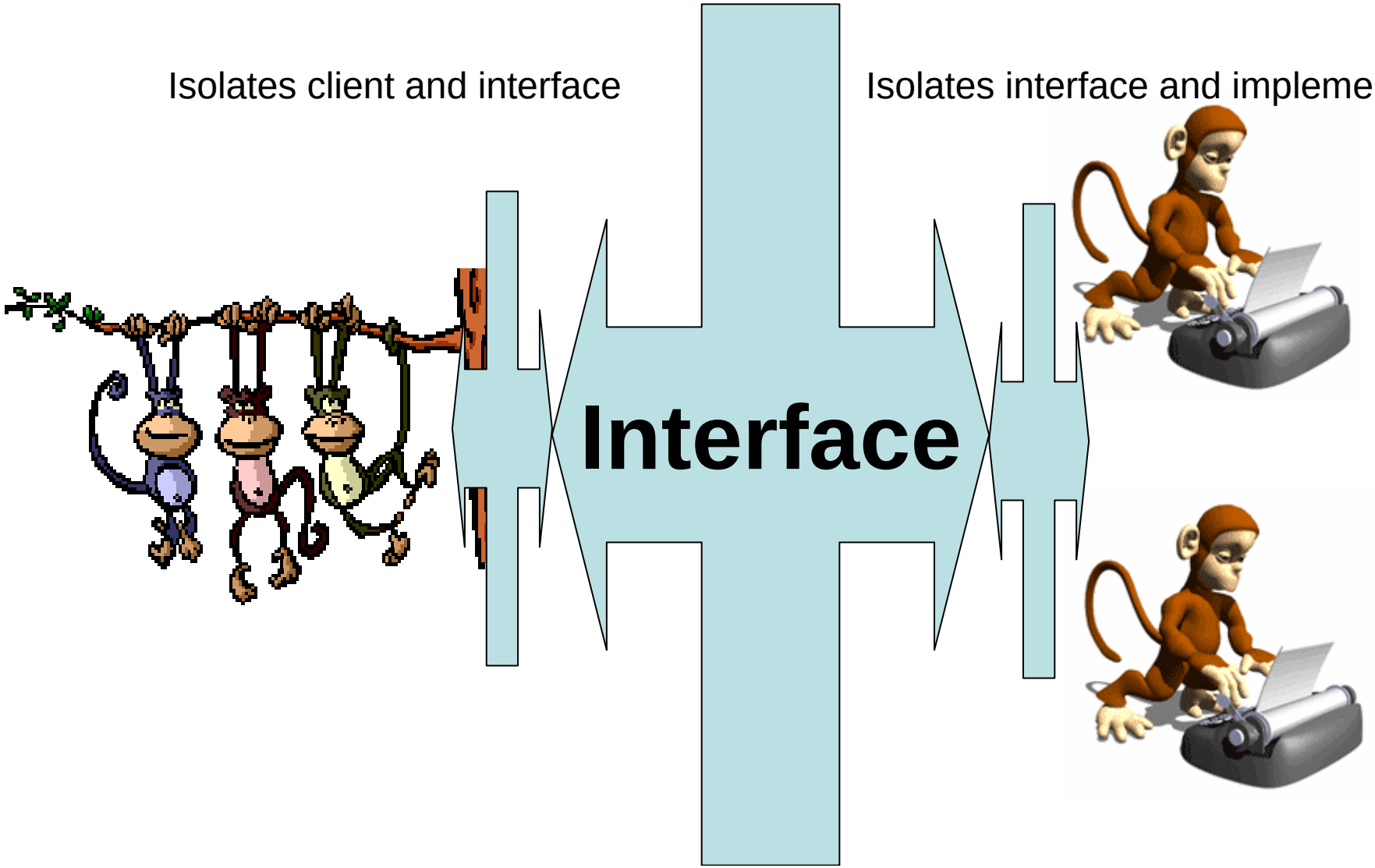
Implementation

Isolates clients …

from implementation

Isolates client and interface

Isolates interface and implementation



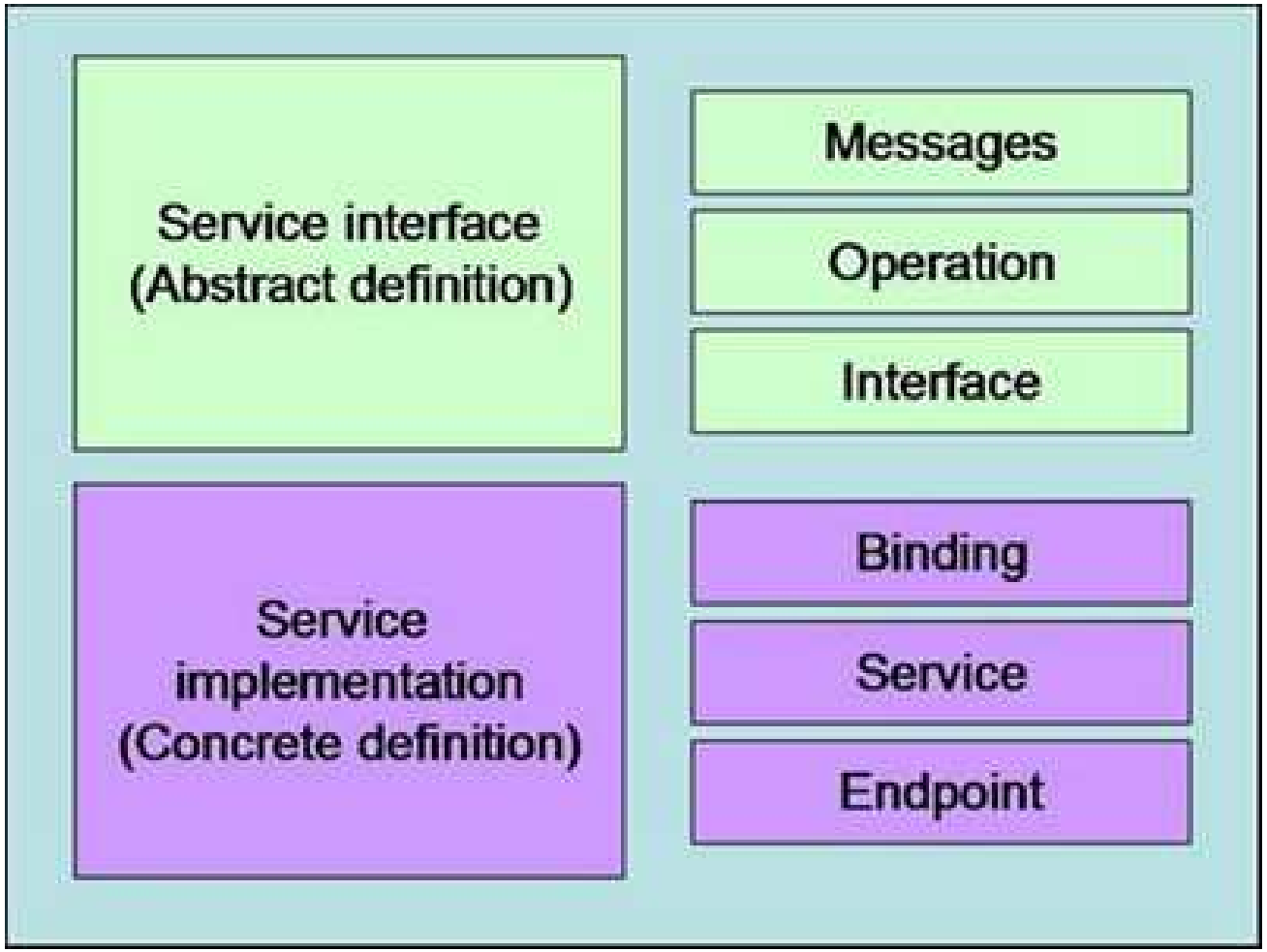
Service Description – critical to SOA lifecycle & roles

- Why? Describe services so
  - Developers/tools can
    - Implement the service
      - Validate the input data
    - Isolate implementation from description
      - Change or switch implementations
      - Hide implementation component model and data types
    - Deploy the service into a hosting environment that supports the protocol/s
    - Advertise the service so that…

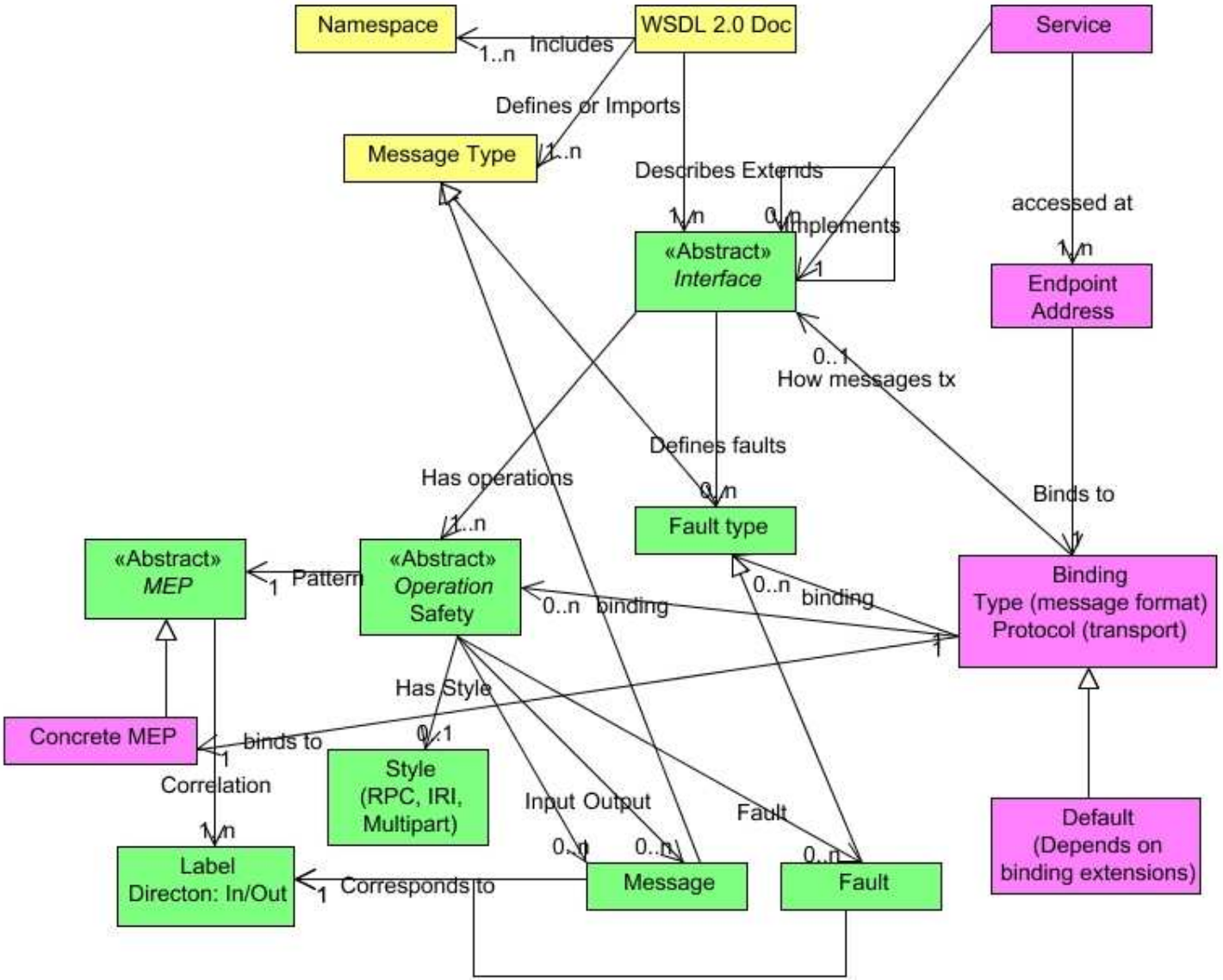
Service Description – critical to SOA lifecycle & roles

- Consumers/tools can
  - Discover them
  - Generate client-side stubs (language <-> protocol mappings)
  - Prepare the input data
  - Locate and invoke them using the agreed protocol
  - Know what to expect, and get an answer back (may not be able to predict which one, or know what it “means” however)
  - Validate returned data
- Not needed?
  - For point-to-point integration (consumer and provider know everything, and share code)
- Where do interfaces come from?
  - But if the interface is independent of implementations and clients, where does the interface come from?
    - What application decomposition modelling process will result in appropriate interfaces? What tools support interface-centric SOA decomposition?
    - Which direction should is proceed? Top-down, bottom-up, middle-out, hybrid, all-at-once, etc?

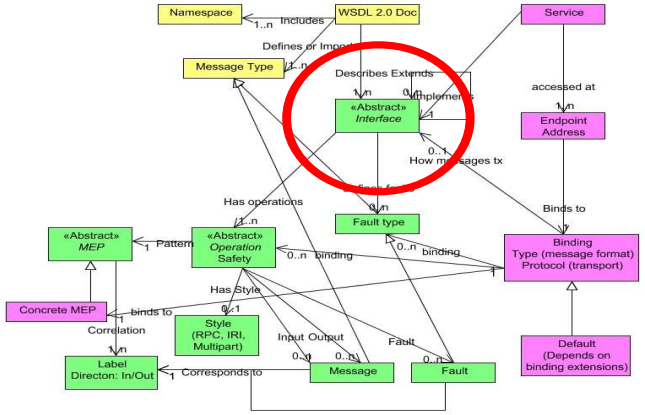
Simple WSDL 2.0 Model



Full Model (approx.)

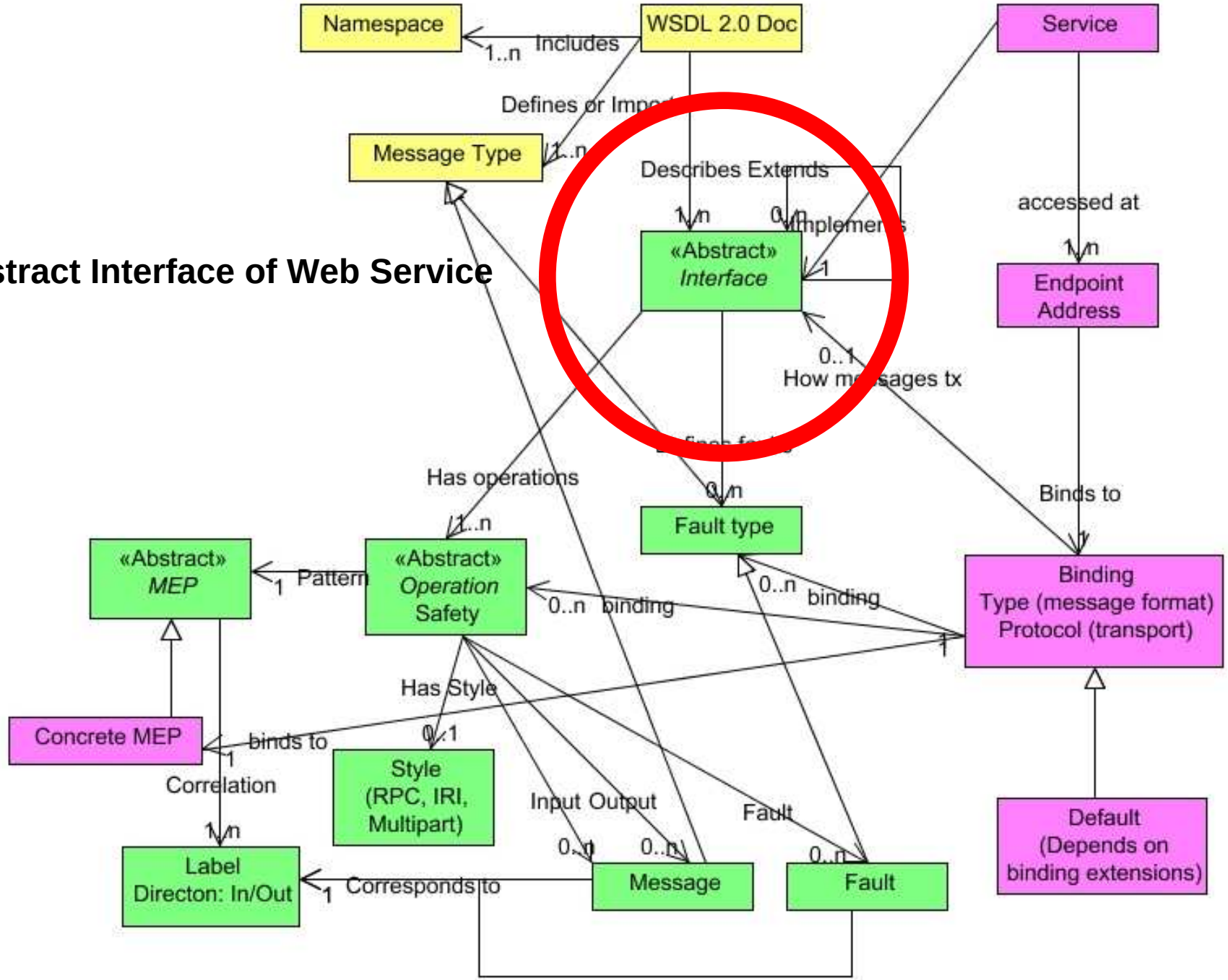


Interfaces



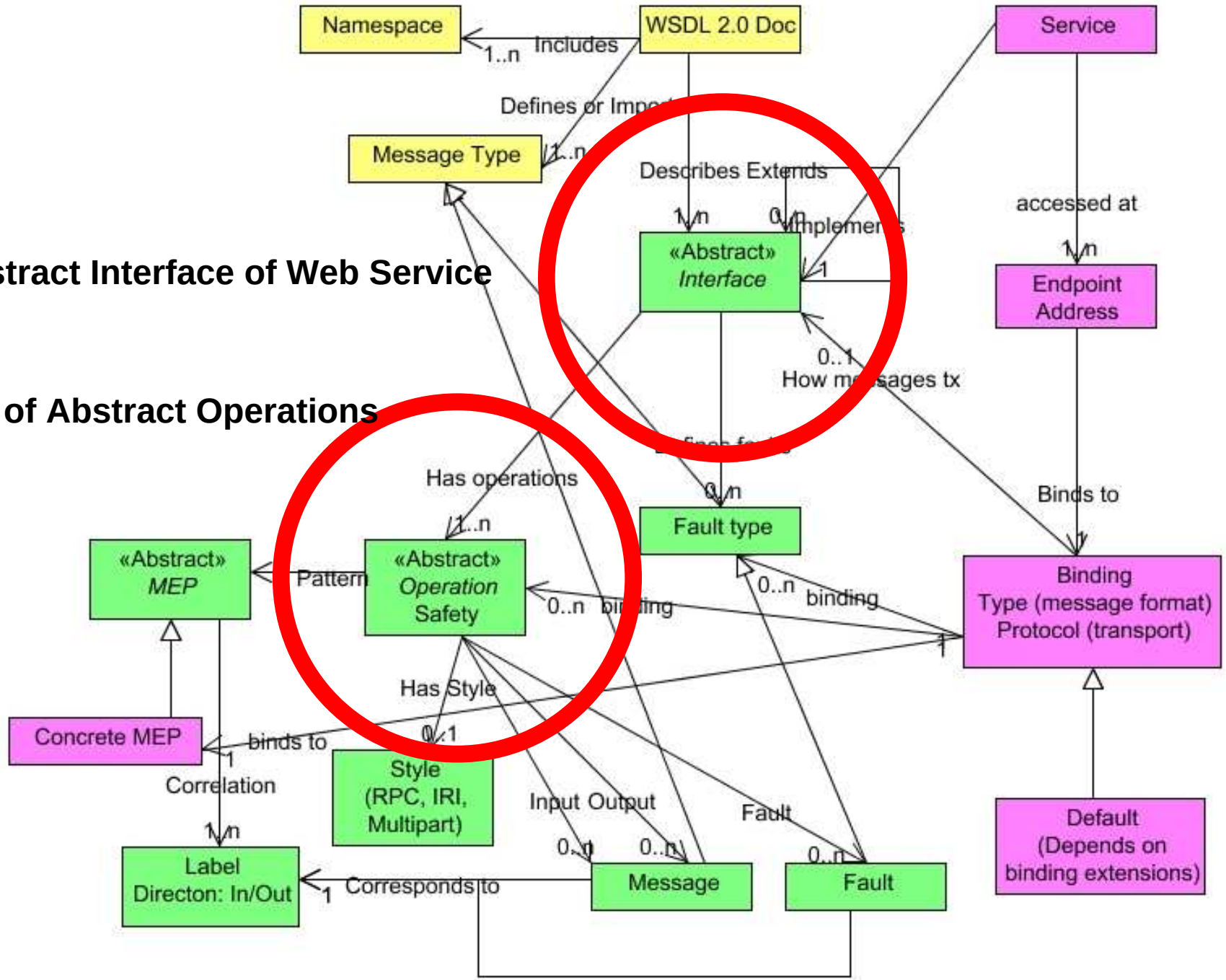
- A WSDL 2.0 interface
  - Defines the abstract interface of a Web service as a set of abstract *operations* and reusable fault messages
  - Can extend other interfaces
- Each operation
  - represents a simple interaction between the client and the service.
  - specifies the types of messages that the service can send or receive as part of that operation (including faults)
  - specifies a message exchange *pattern* that indicates the sequence in which the associated messages are to be transmitted between the parties.
  - Has a style and safety

Abstract Interface of Web Service



Abstract Interface of Web Service

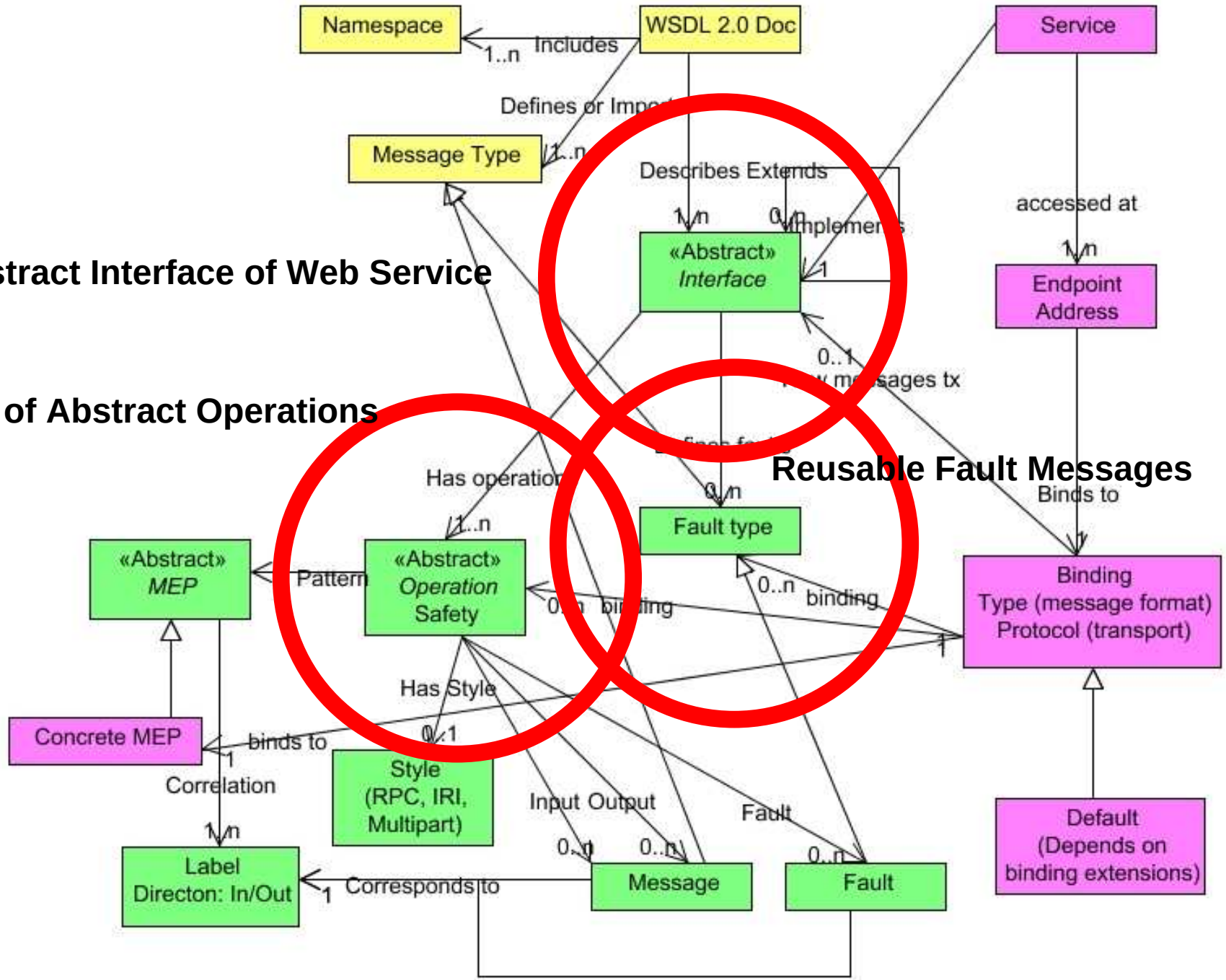
Set of Abstract Operations

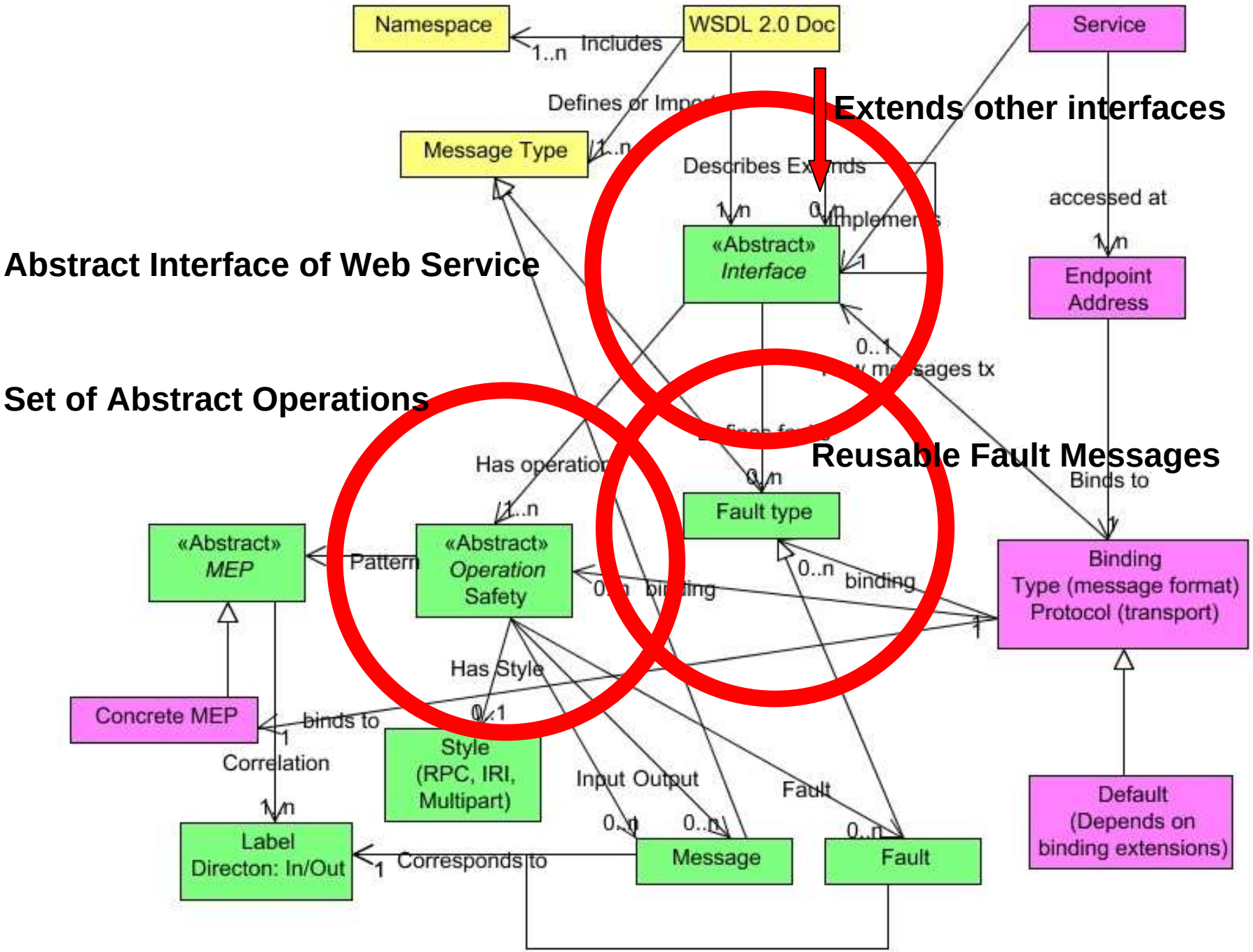


Abstract Interface of Web Service

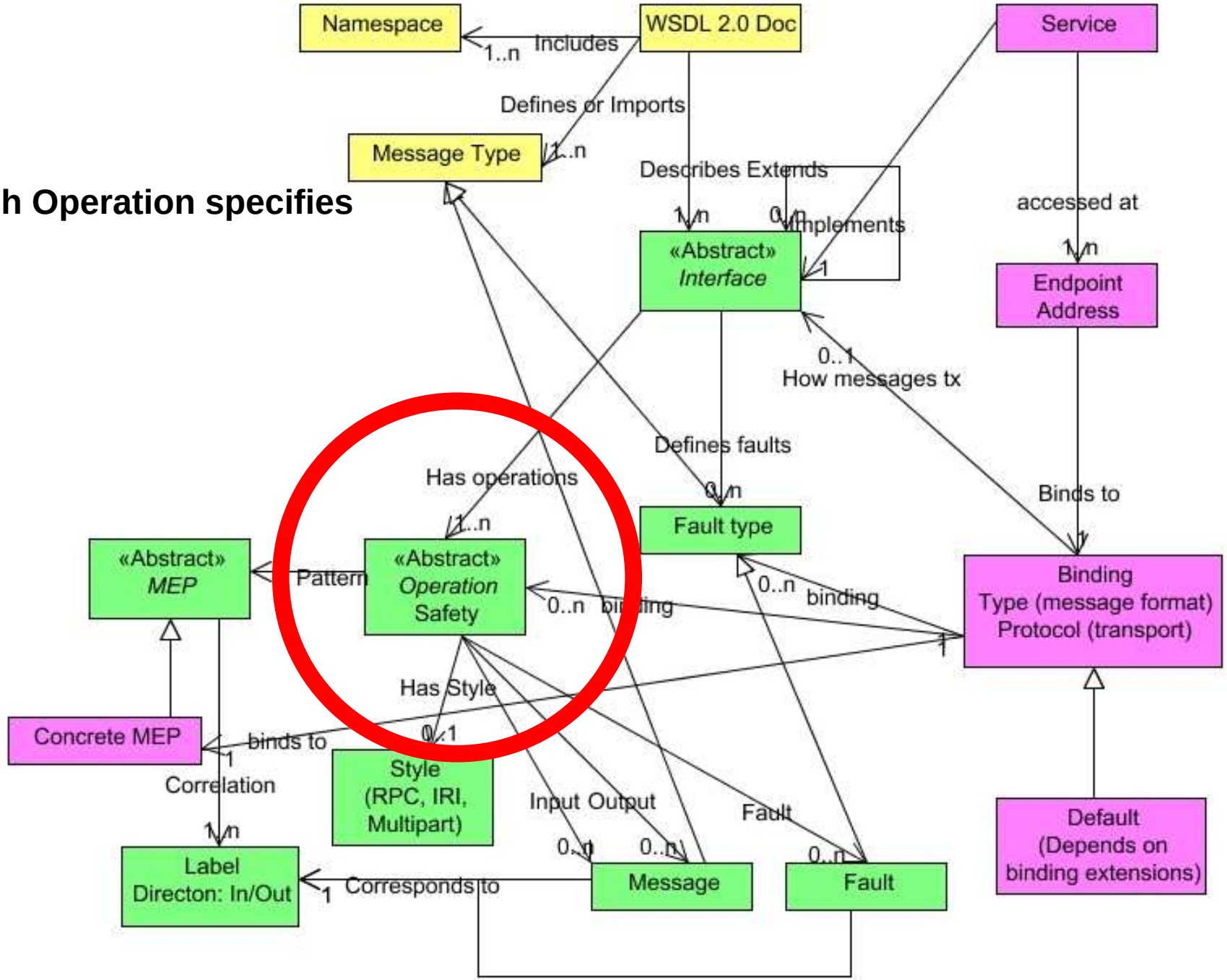
Set of Abstract Operations

Reusable Fault Messages

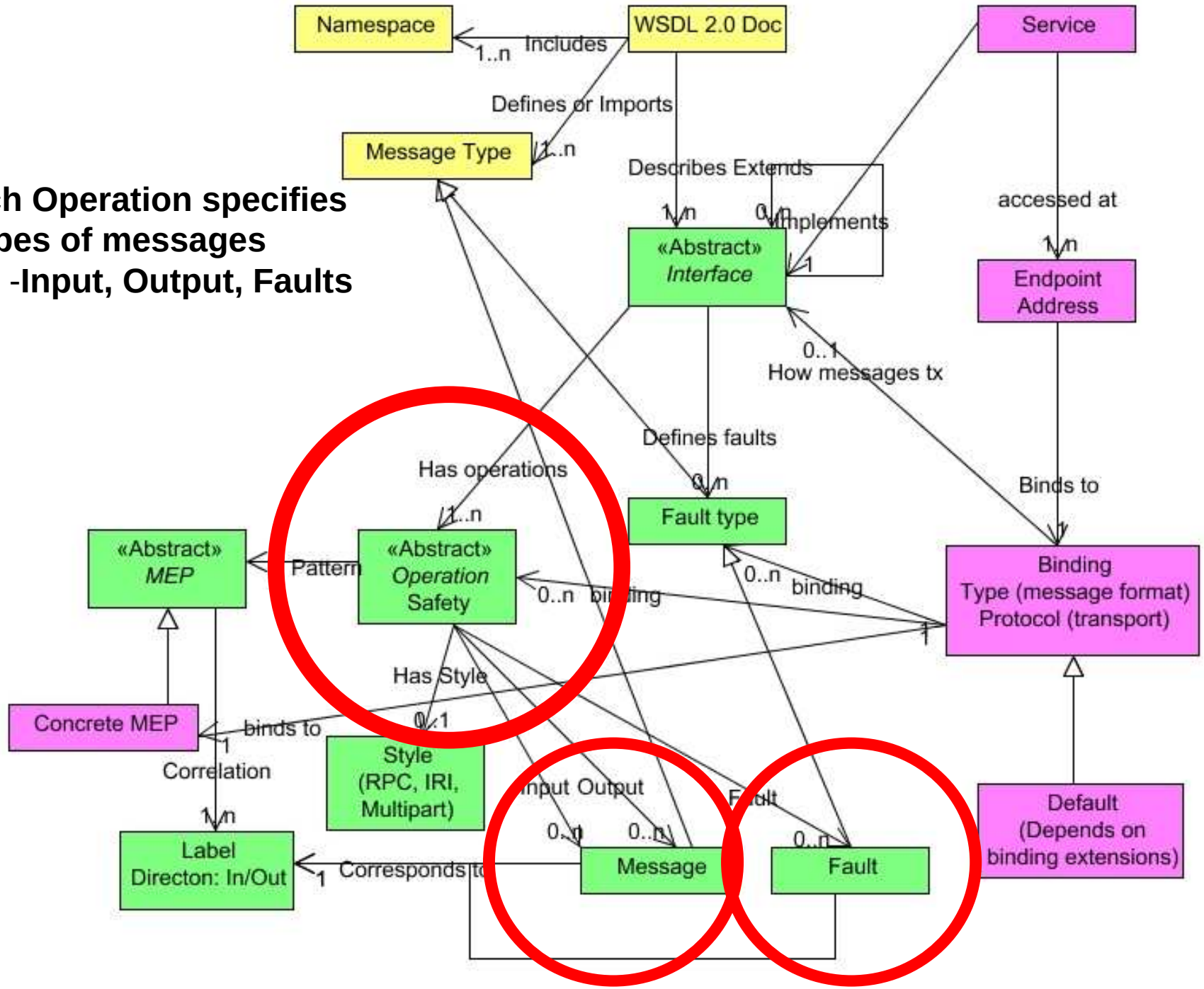



Namespace
WSDL 2.0 Doc
Service
Includes
1..n
Extends other interfaces
Message Type
Describes
accessed at
«Abstract»
Interface
Endpoint
Address
Abstract Interface of Web Service
Set of Abstract Operations
Reusable Fault Messages
Has operation
Fault type
Binds to
«Abstract»
MEP
Pattern
«Abstract»
Operation
Safety
binding
Binding
Type (message format)
Protocol (transport)
Has Style
Concrete MEP
binds to
Correlation
Style
(RPC, IRI,
Multipart)
Input Output
Fault
Default
(Depends on
binding extensions)
Label
Directon: In/Out
Corresponds to
Message
Fault

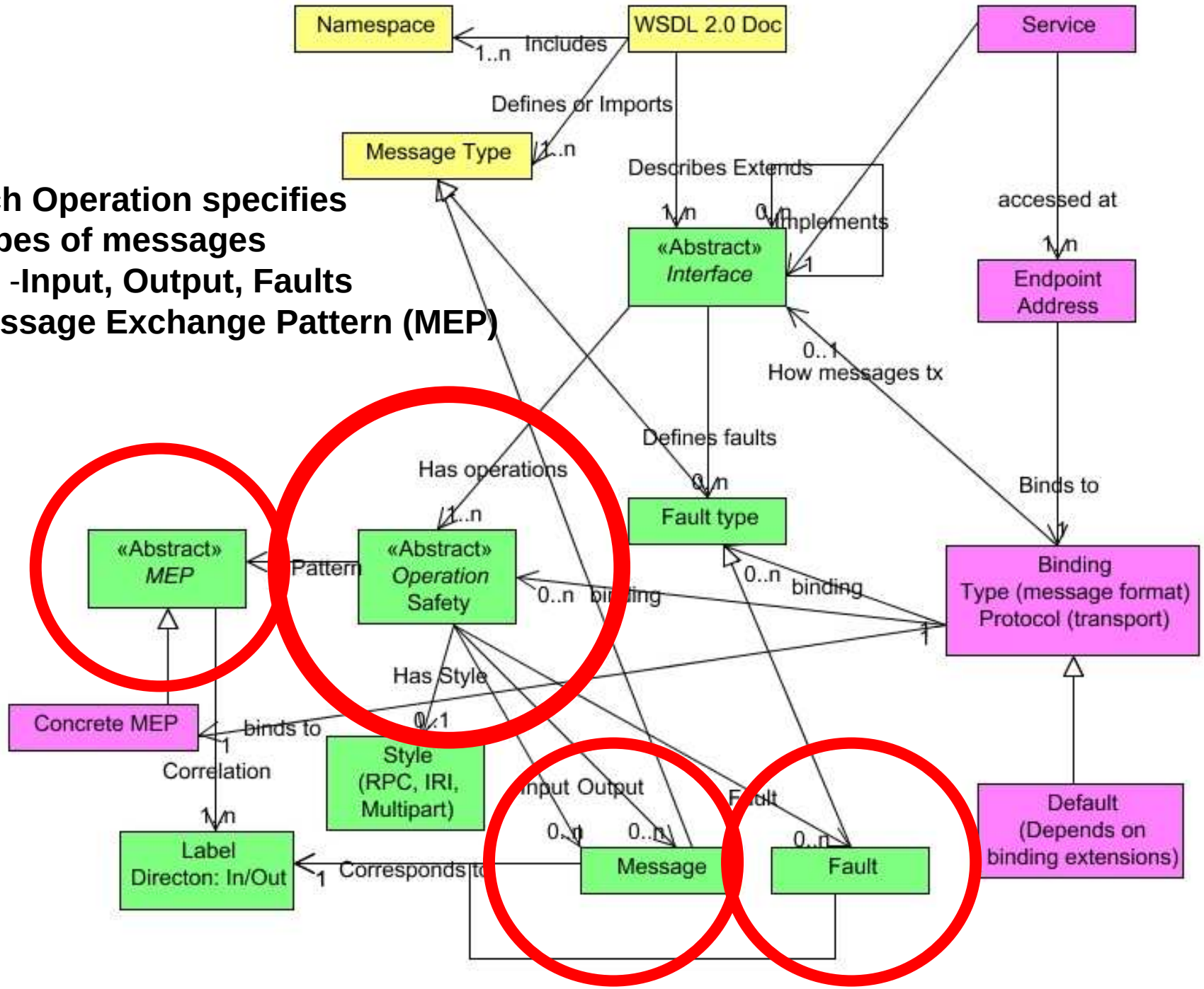
Each Operation specifies



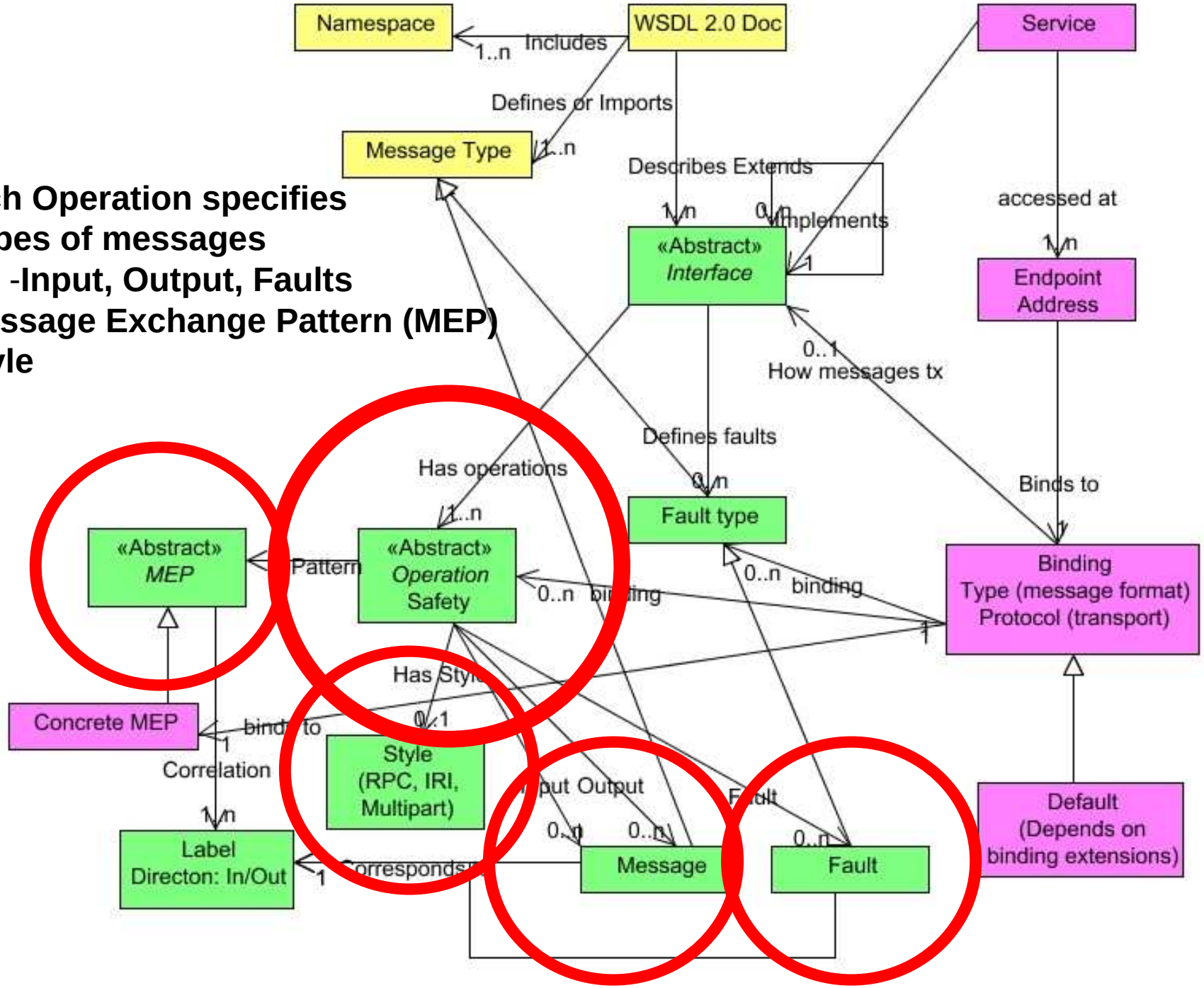
Each Operation specifies
-Types of messages
-Input, Output, Faults



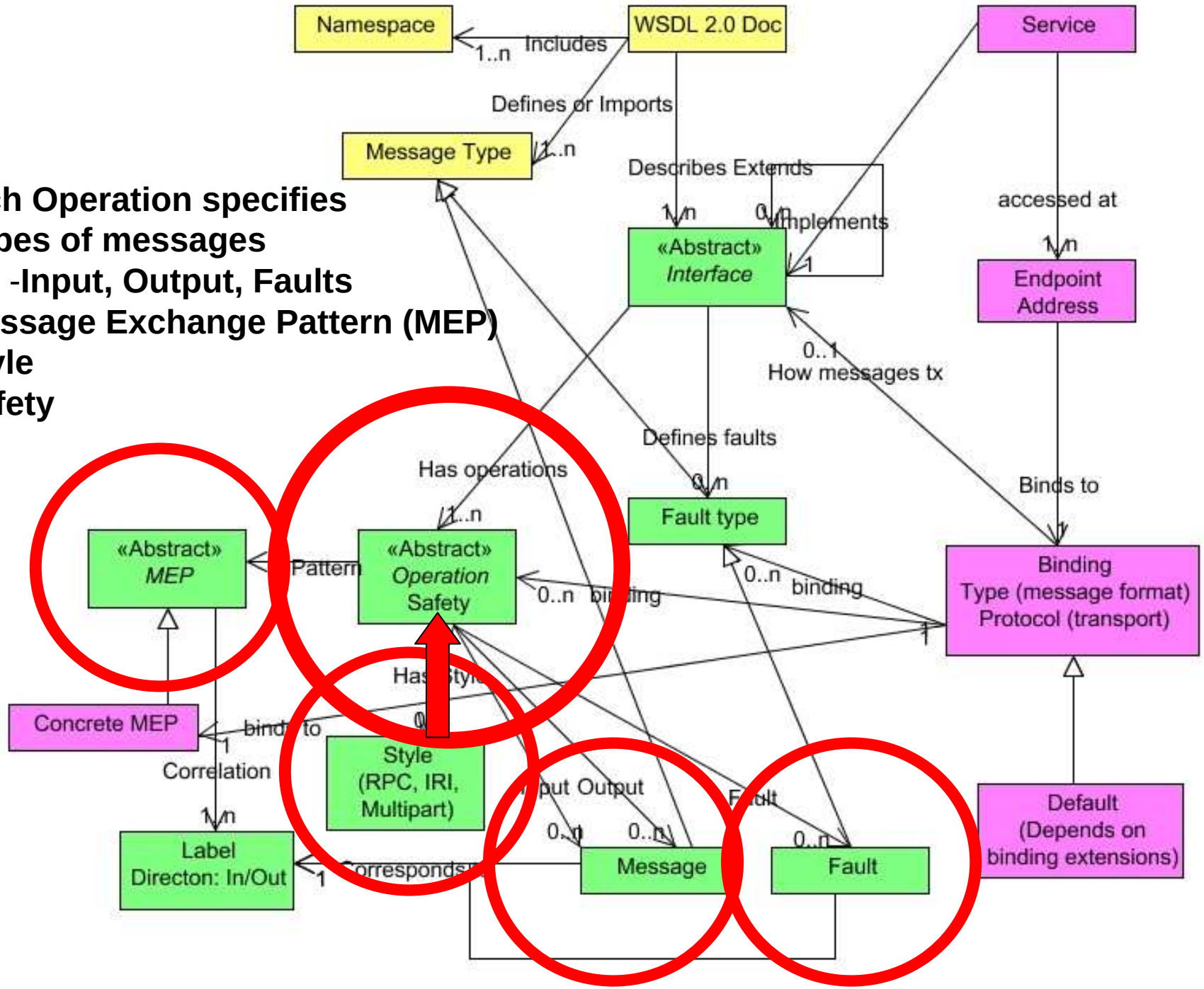
Each Operation specifies
-Types of messages
-Input, Output, Faults
-Message Exchange Pattern (MEP)



Each Operation specifies
-Types of messages
-Input, Output, Faults
-Message Exchange Pattern (MEP)
-Style



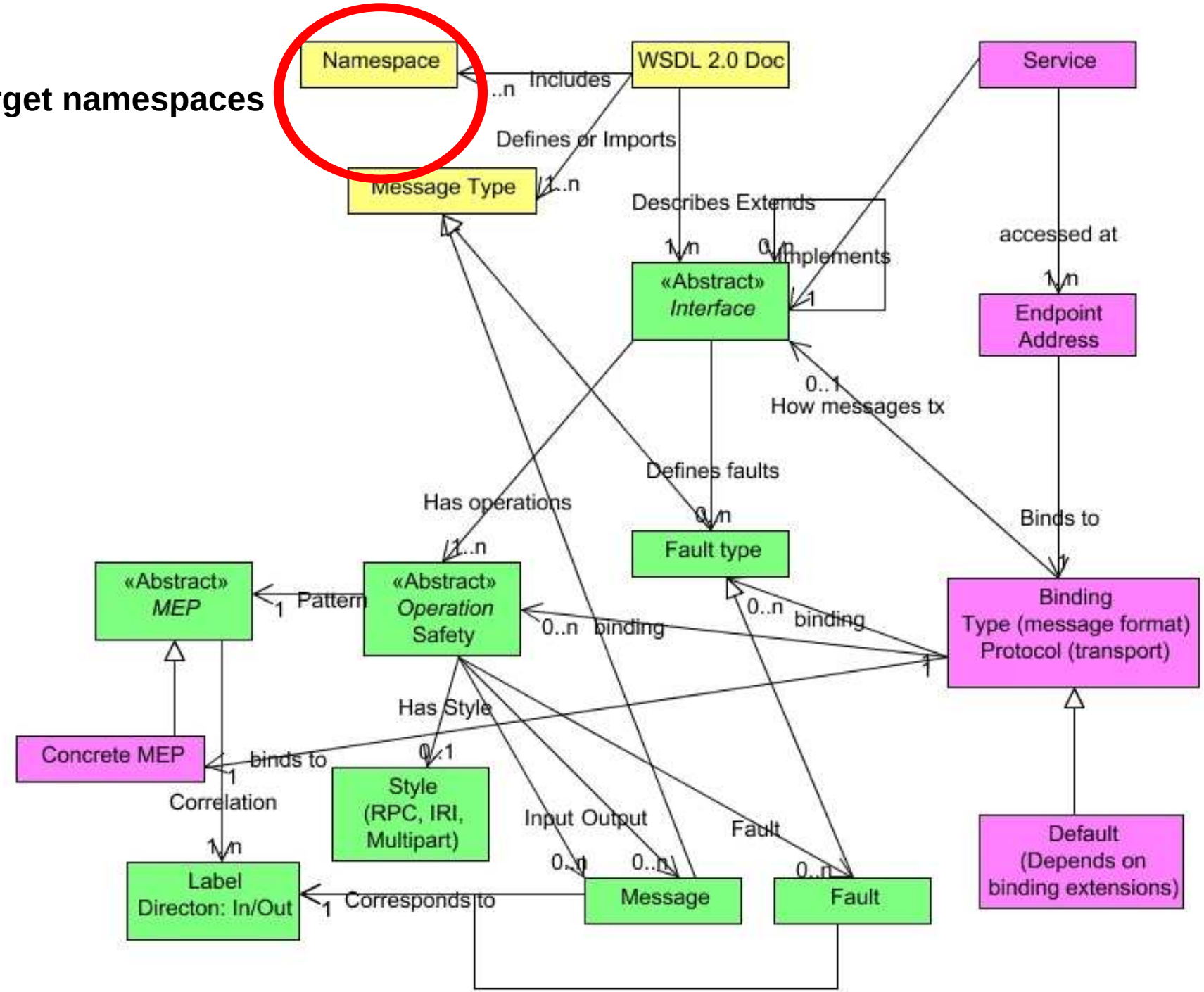
Each Operation specifies
-Types of messages
-Input, Output, Faults
-Message Exchange Pattern (MEP)
-Style
-Safety



Message Types

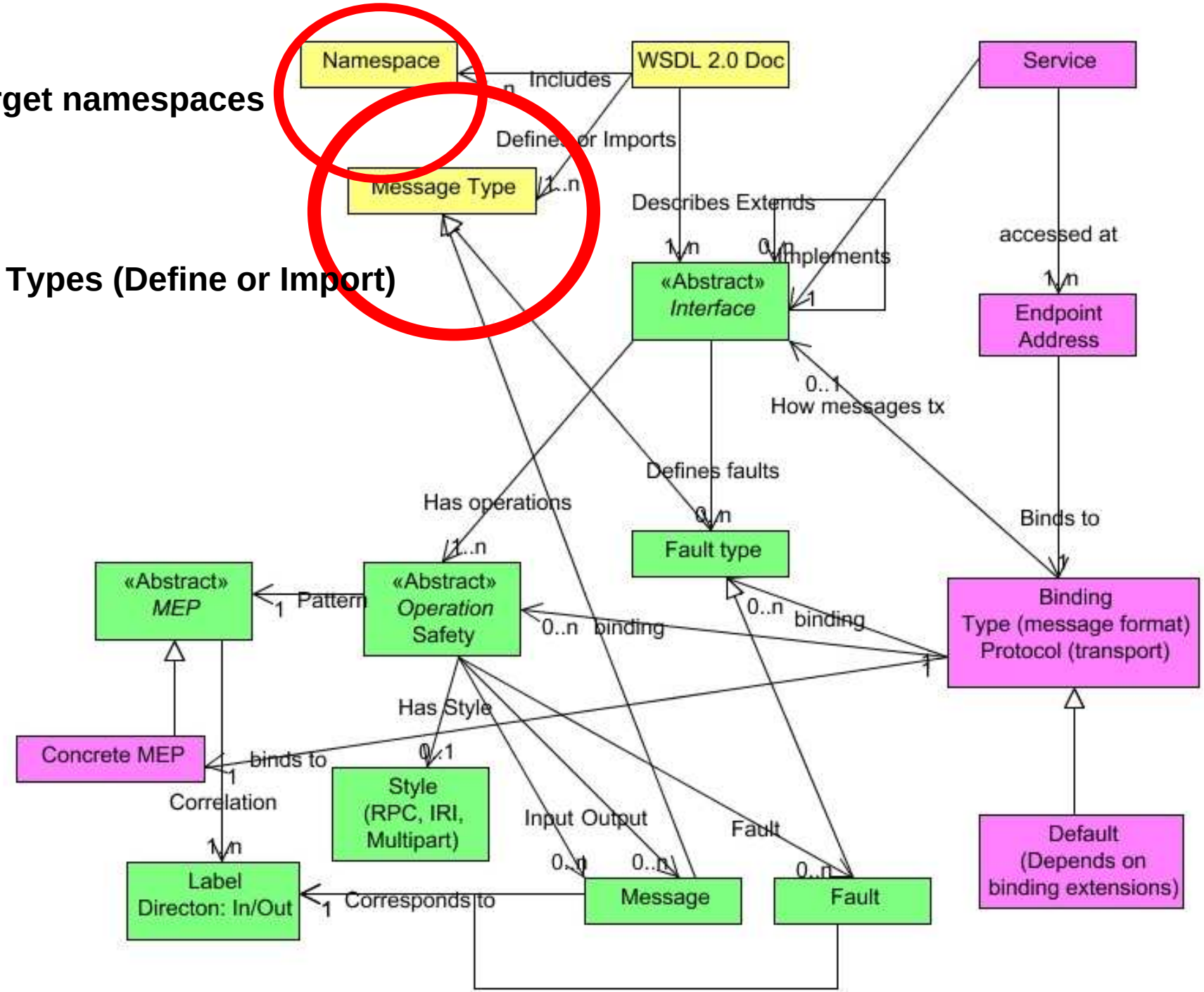
- Top-level WSDL document **description** describes
  - The target namespaces
  - Documentation (comments)
  - Types
    - Define the type of every message
      - Including faults
    - Multiple solutions
      - Minimally, XML Schema supported
      - Define message types
        - » In WSDL document, or
        - » Import them

Target namespaces



Target namespaces

Types (Define or Import)



Description example

```
<?xml version="1.0" encoding="utf-8" ?>
<description xmlns=http://www.w3.org/2006/01/wsdl
   targetNamespace= http://greath.example.com/2004/wsdl/resSvc
   xmlns:tns= http://greath.example.com/2004/wsdl/resSvc
   xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc" . . . > ...
   <types>
   <xs:schema
        xmlns:xs=http://www.w3.org/2001/XMLSchema
        targetNamespace="http://greath.example.com/2004/schemas/resSvc"
        xmlns="http://greath.example.com/2004/schemas/resSvc">
        <xs:element name="checkAvailability" type="tCheckAvailability"/>
        <xs:complexType name="tCheckAvailability">
                <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> . . .
</description>
```

Interface Example

```
<types> ...
</types>

<interface name = "reservationInterface" >
  <fault name = "invalidDataFault" element =
      "ghns:invalidDataError"/>
  <operation name="opCheckAvailability"
  pattern="http://www.w3.org/2006/01/wsdl/in-out"
  style="http://www.w3.org/2006/01/wsdl/style/iri"
  wsdlx:safe = "true">
  <input messageLabel="In“
      element="ghns:checkAvailability" />
  <output messageLabel="Out"
      element="ghns:checkAvailabilityResponse" />
  <outfault ref="tns:invalidDataFault"
      messageLabel="Out"/> </operation>
</interface>
```

Interface Example

```
<types> ...
</types>

<interface name = "reservationInterface" >
  <fault name = "invalidDataFault" element =
      "ghns:invalidDataError"/>
  <operation name="opCheckAvailability"
  pattern="http://www.w3.org/2006/01/wsdl/in-out"
  style="http:
  wsdlx:safe =
  <input messa
      element="ghns:checkAvailability" />
  <output messageLabel="Out"
      element="ghns:checkAvailabilityResponse" />
  <outfault ref="tns:invalidDataFault"
      messageLabel="Out"/> </operation>
</interface>
```

URI for In-Out MEP

Interface Example

```
<types> ...
</types>

<interface name = "reservationInterface" >
  <fault name = "invalidDataFault" element =
      "ghns:invalidDataError"/>
  <operation name="opChe
  pattern="http://www.w3
  style="http://www.w3.o
  wsdlx:safe = "true">
  <input messageLabel="I
      element="ghns:chec
  <output messageLabel="Out"
      element="ghns:checkAvailabilityResponse" />
  <outfault ref="tns:invalidDataFault"
      messageLabel="Out"/> </operation>
</interface>
```

This operation will not obligate the client in any way (“false” means don’t know or there will be an obligation).

Interface Example

```
<types>
</types>
```

a message exchange pattern represents a template for a message sequence – potentially multiple input and output messages

```
<interface name = "reservationInterface" >
   <fault name = "invalidDataFault" element =
       "ghns:invalidDataError"/>
   <operatio
   pattern="
   style="ht
   wsdlx:saf
   <input messageLabel="In“
       element="ghns:checkAvailability" />
   <output messageLabel="Out"
       element="ghns:checkAvailabilityResponse" />
   <outfault ref="tns:invalidDataFault"
       messageLabel="Out"/> </operation>
</interface>
```

Labels indicate which input message in the pattern this particular input message represents

Interface Example

```
<types> ...
</types>

<interface name = "reservationInterface" >
  <fault name = "invalidDataFault" element =
      "ghns:invalidDataError"/>
  <operation name="onCheckAvailability"
  pattern="h                                    sdl/in-out"
  style="htt                                    l/style/iri"
  wsdlx:safe
  <input mes
      elemen                                    />
  <output messageLabel="Out"
      element="ghns:checkAvailabilityResponse" />
  <outfault ref="tns:invalidDataFault"
      messageLabel="Out"/> </operation>
</interface>
```

name of a previously defined fault in this interface

Interface Example

```
<types> ...
</types>

<interface name = "reservationInterface" >
  <fault name = "invalidDataFault" element =
      "ghns:invalidDataError"/>
  <operation name="opCheckAvailability"
  pattern="http://www.w3.org/2006/01/wsdl/in-out"
  style="http://www.w3.org/2006/01/wsdl/style/iri"
  wsdlx:safe = "true">
  <input messageLabel="In“
      element="ghns:checkAvailability" />
  <output messageLabel="Out"
      element="ghns:checkAvailabilityResponse" />
  <outfault ref="tns:invalidDataFault"
      messageLabel="Out"/> </operation>
</interface>
```

Message associated with the fault

MEPs

- pattern="http://www.w3.org/2006/01/wsdl/in-out"
  - This line specifies that this operation will use the in-out pattern.
  - WSDL 2.0 uses URIs to identify message exchange patterns in order to ensure that the identifiers are globally unambiguous, while also permitting future new patterns to be defined by anyone (extensions).

Safety

- wsdlx:safe="true" >
  - This line indicates that this operation will not obligate the client in any way (“false” means don’t know or there will be an obligation).

MEPs (and messages)

- <input messageLabel="In"
  - The input element specifies an input message. Even though we have already specified which message exchange pattern the operation will use, a message exchange pattern represents a template for a message sequence, and in theory could consist of multiple input and/or output messages. Thus we must also indicate which potential input message in the pattern this particular input message represents. This is the purpose of the messageLabel attribute. Since the in-out pattern that we've chosen to use only has one input message, it is trivial in this case: we simply fill in the message label "In" that was defined in *WSDL 2.0 Predefined Extensions* [*WSDL 2.0 Adjuncts*] section 2.2.3 In-Out for the in-out pattern. However, if a new pattern is defined that involve multiple input messages, then the different input messages in the pattern could then be distinguished by using different labels.

Faults

- <outfault ref="tns:invalidDataFault" messageLabel="Out"/>
  - This associates an output fault with this operation. Faults are declared a little differently than normal messages. The ref attribute refers to the name of a previously defined fault in this interface -- not a message schema type directly. Since message exchange patterns could in general involve a sequence of several messages, a fault could potentially occur at various points within the message sequence. Because one may wish to associate a different fault with each permitted point in the sequence, the messageLabel is used to indicate the desired point for this particular fault. It does so indirectly by specifying the message that will either trigger this fault or that this fault will replace, depending on the pattern.

Interface inheritance

- An interface can extend or inherit from one or more other interfaces
  - the interface contains the operations it defines directly, and
  - the operations of the interfaces it extends
- Inheritance loops prohibited
  - the interfaces that a given interface extends must NOT themselves extend that interface either directly or indirectly.
- What happens when operations from two different interfaces have the same target namespace and operation name?
  - If the component models are “the same” (see spec) then they are considered to be the same operation (collapsed)
  - Otherwise an error.
- Cool things
  - Reuse of interface definitions, and
  - Backwards compatibility
    - A client that understands only the extended interface(s) will still work with an interface that extends them.
  - Interface inheritance is one reason that the OGSA/Grid community developed their own GWSDL – now they could use WSDL 2.0 - in theory.

Interface Extension Example

An out-only, logging operation

```
<interface name = "messageLogInterface" >
   <operation name="opLogMessage"
   pattern="http://www.w3.org/2006/01/wsdl/out-only">
   <output messageLabel="out"
   element="ghns:messageLog" /> </operation>
</interface>
```

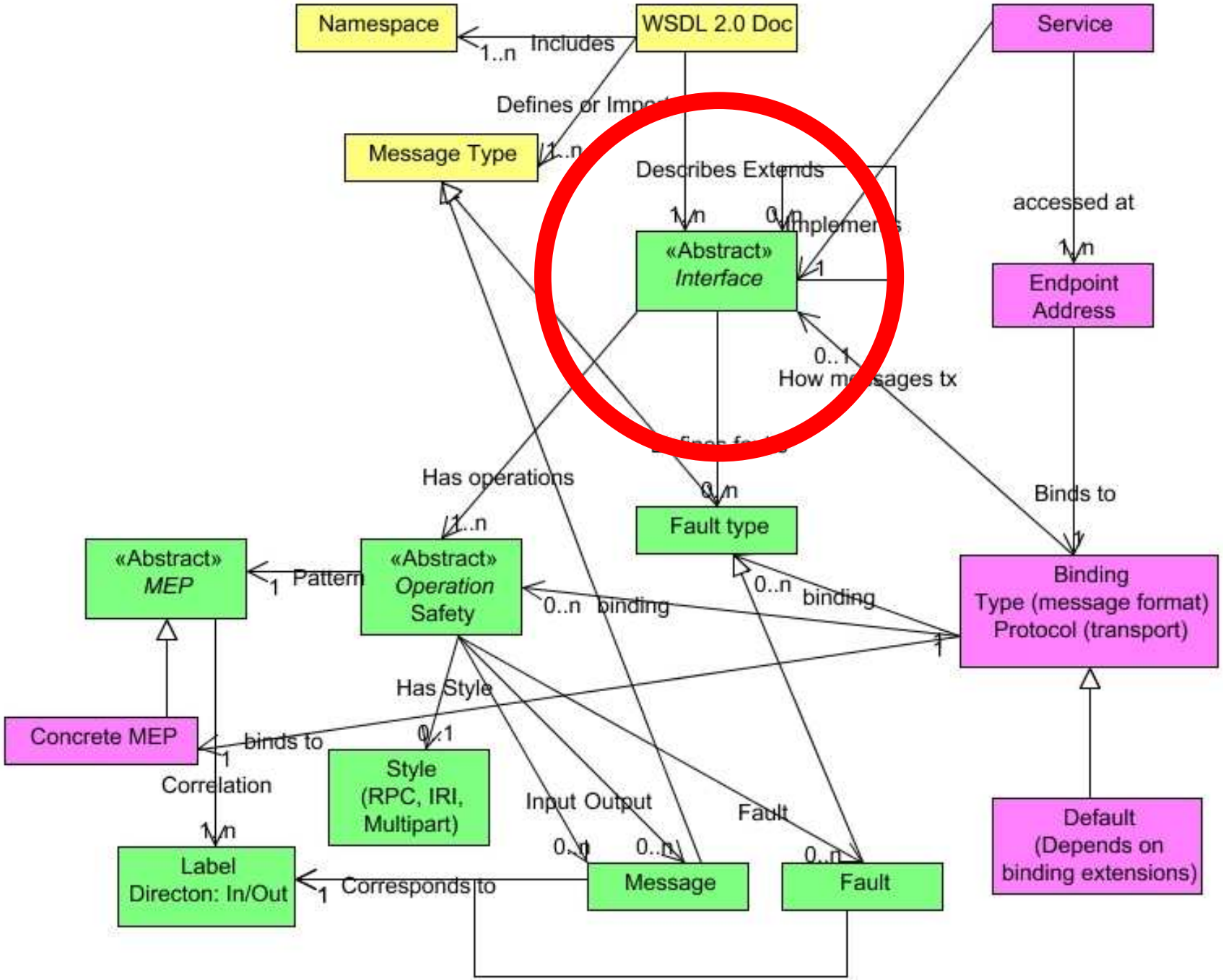
```
<interface name="reservationInterface"
   extends="tns:messageLogInterface" >
   <operation name="opCheckAvailability"
   . . . (as before)
   </operation>
</interface>
```

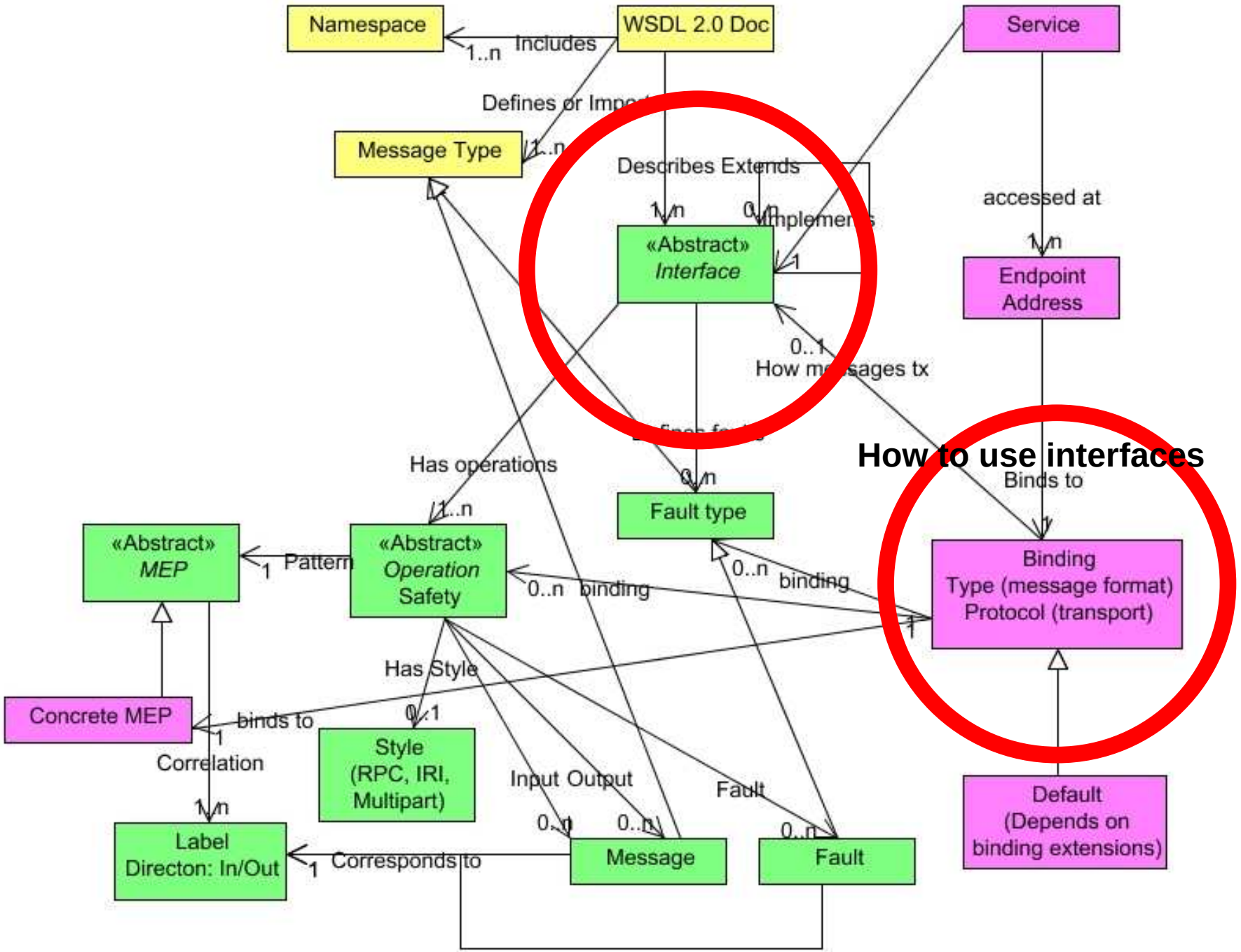
Extended by reservationInterface, which now has two operations

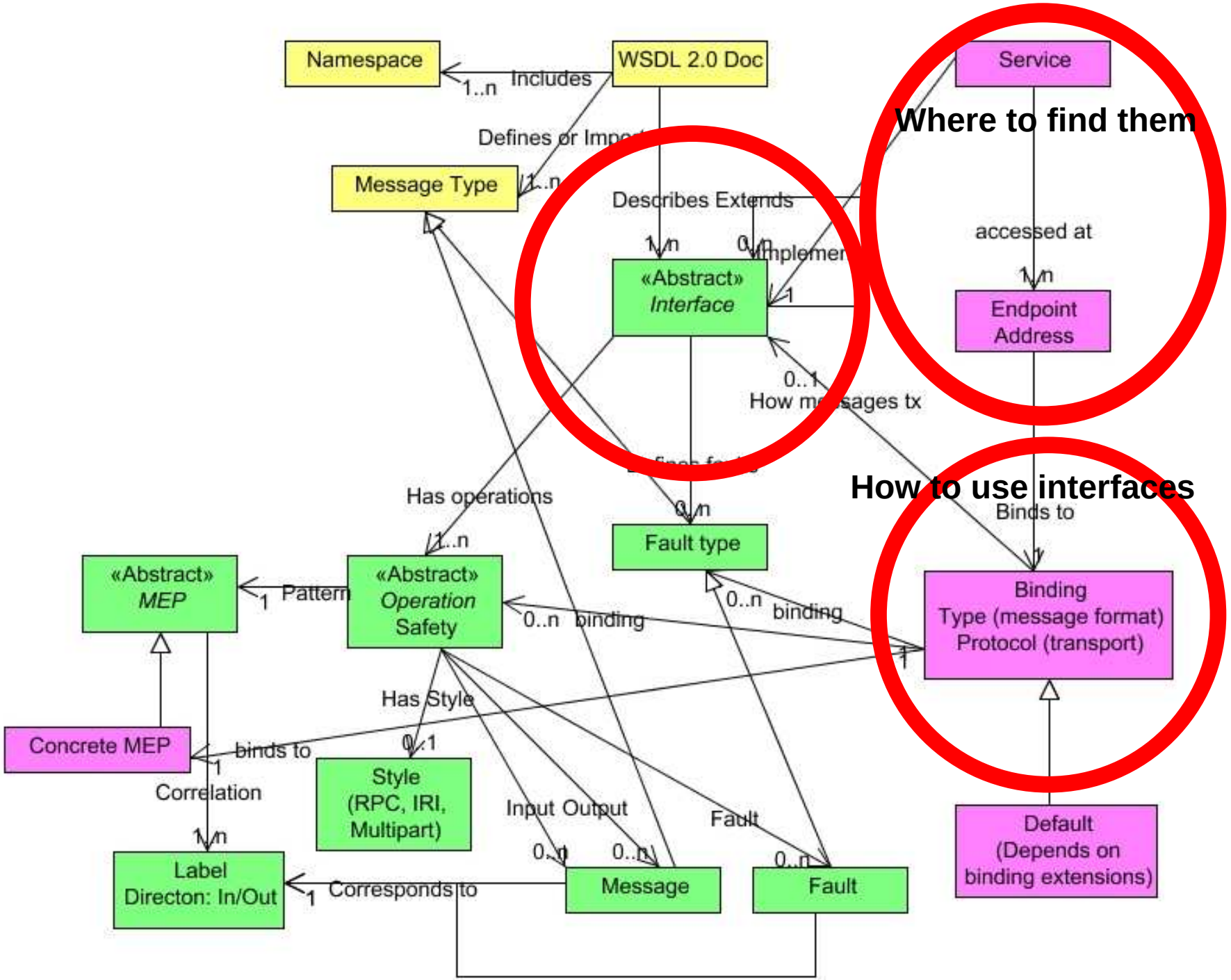
Time to get concrete

What’s missing? Concrete.

- Interfaces are only abstract
- In order to use them need to know
  - *How* to use them (how messages will be exchanged)
    - Bindings
  - *Where* to find them
    - Service


Namespace
WSDL 2.0 Doc
Service
Includes
1..n
Defines or Imports
Message Type
1..n
Describes Extends
1..n
0..n
Implements
accessed at
1..n
«Abstract»
Interface
1
Endpoint
Address
0..1
How messages tx
Has operations
1..n
0..n
Fault type
Binds to
«Abstract»
MEP
1
Pattern
«Abstract»
Operation
Safety
0..n
binding
0..n
binding
1
Binding
Type (message format)
Protocol (transport)
Has Style
Concrete MEP
binds to
1
0..1
Style
(RPC, IRI,
Multipart)
Correlation
1..n
Input Output
Fault
0..n
0..n
0..n
Label
Directon: In/Out
1
Corresponds to
Message
Fault
Default
(Depends on
binding extensions)

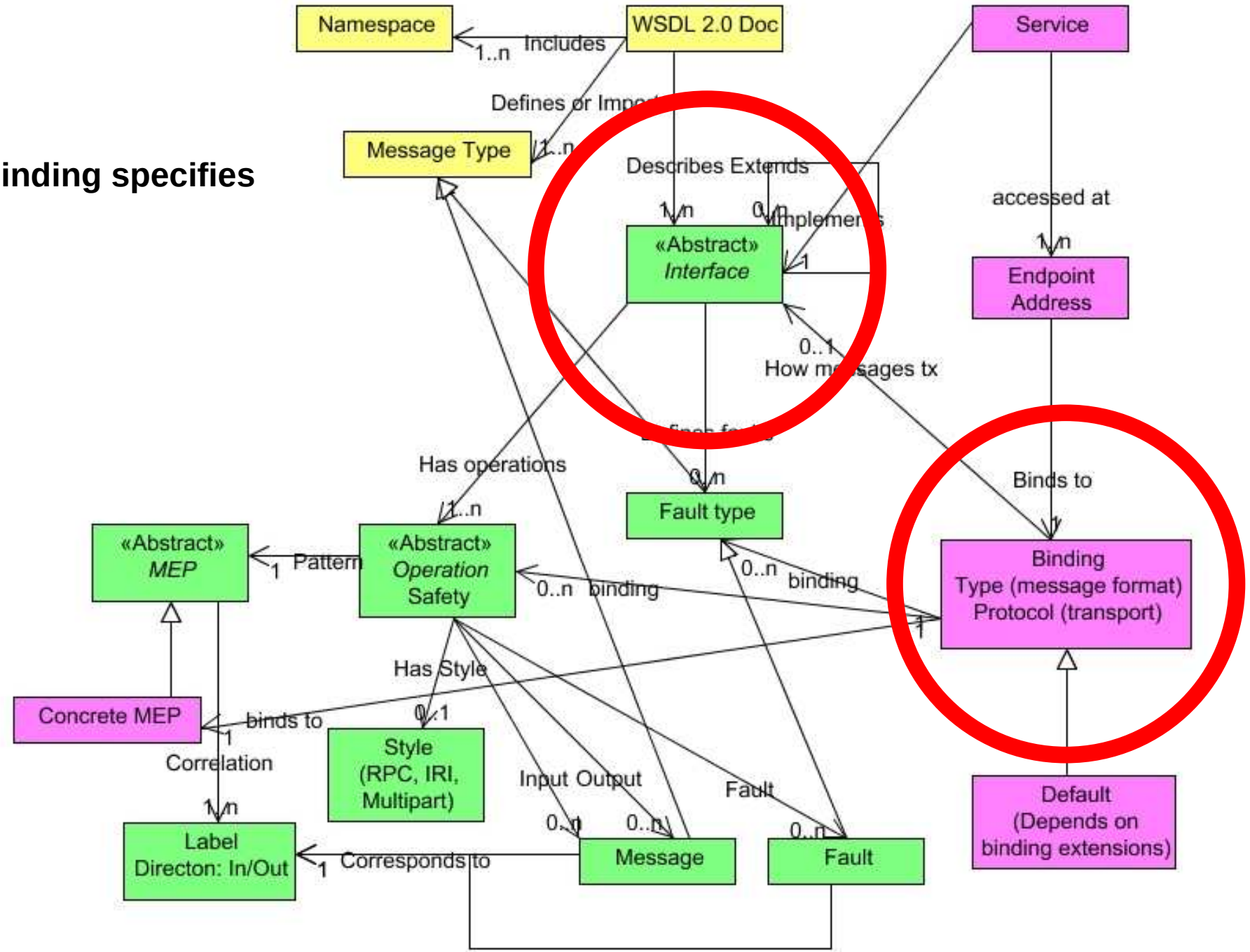

Namespace
WSDL 2.0 Doc
Service
Includes
1..n
Defines or Imports
Message Type
Describes Extends
«Abstract»
Interface
Implements
accessed at
Endpoint
Address
How messages tx
Has operations
Fault type
Binds to
«Abstract»
MEP
Pattern
«Abstract»
Operation
Safety
binding
Binding
Type (message format)
Protocol (transport)
Has Style
Concrete MEP
binds to
Correlation
Style
(RPC, IRI,
Multipart)
Input Output
Fault
Default
(Depends on
binding extensions)
Label
Directon: In/Out
Corresponds to
Message
Fault
How to use interfaces


Namespace
WSDL 2.0 Doc
Includes
1..n
Defines or Impo
Message Type
1..n
Describes Extends
1..n
0..n
«Abstract»
Interface
1
Service
Where to find them
accessed at
1..n
Endpoint
Address
0..1
How me sages tx
Has operations
1..n
0..n
Fault type
How to use interfaces
Binds to
«Abstract»
MEP
1
Pattern
«Abstract»
Operation
Safety
0..n
binding
0..n
binding
Binding
Type (message format)
Protocol (transport)
Has Style
0..1
Concrete MEP
binds to
1
Correlation
1..n
Style
(RPC, IRI,
Multipart)
Input Output
0..n
0..n
Fault
0..n
Default
(Depends on
binding extensions)
Label
Directon: In/Out
1
Corresponds to
Message
Fault

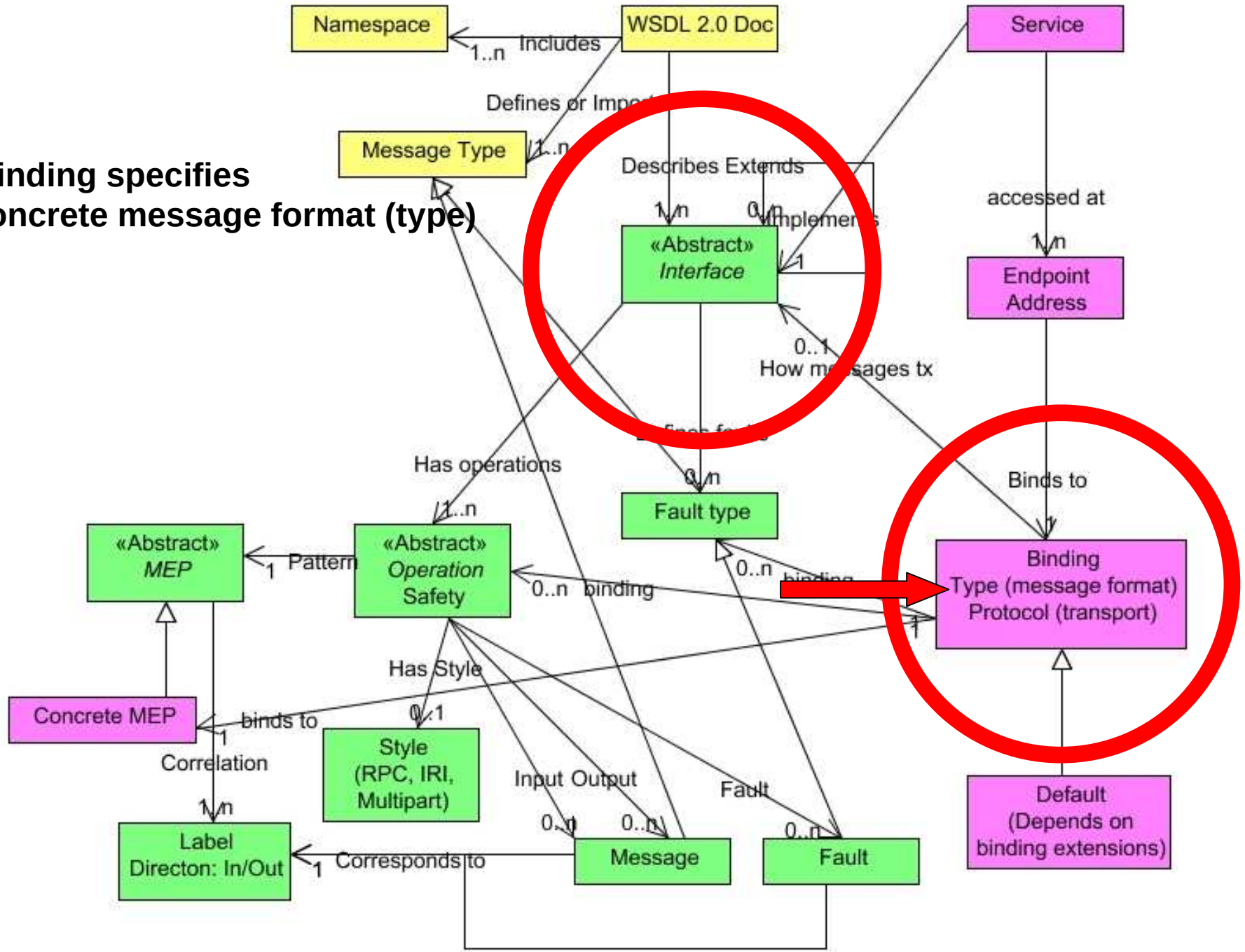
Binding

- A binding specifies
  - concrete message format (E.g. SOAP 1.1, 1.2) and
  - transmission protocol details (E.g. HTTP, HTTPS)
  - Concrete MEP
- Every operation and fault in the interface must have a binding.
- Default bindings
  - if the “binding extension” (see spec: WSDL 2.0 extensions) provides suitable defaulting rules, then
    - Bindings only needed at the interface level
    - will implicitly propagate bindings to operations of the interface.

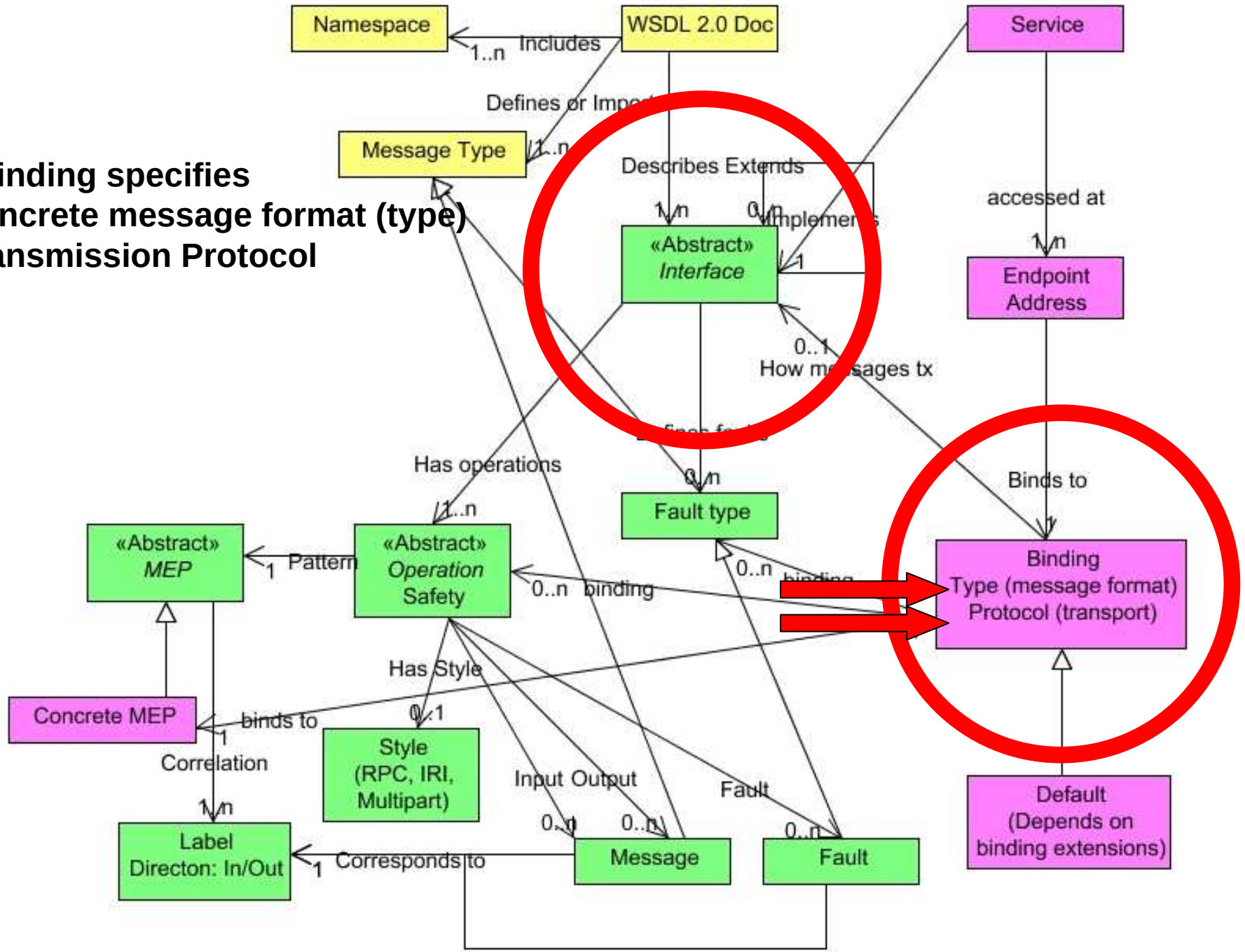
A binding specifies



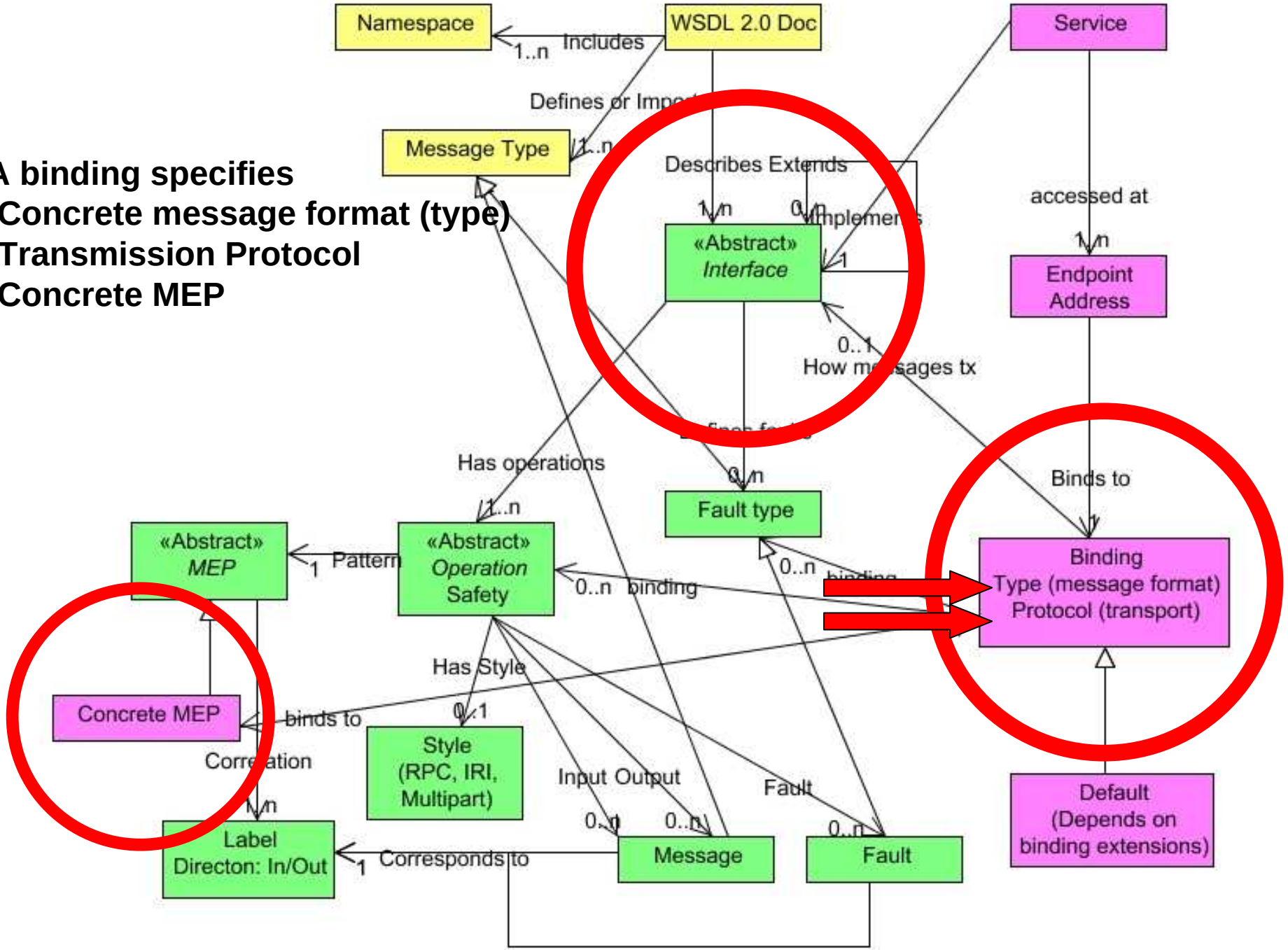
A binding specifies
- Concrete message format (type)



A binding specifies
-Concrete message format (type)
-Transmission Protocol

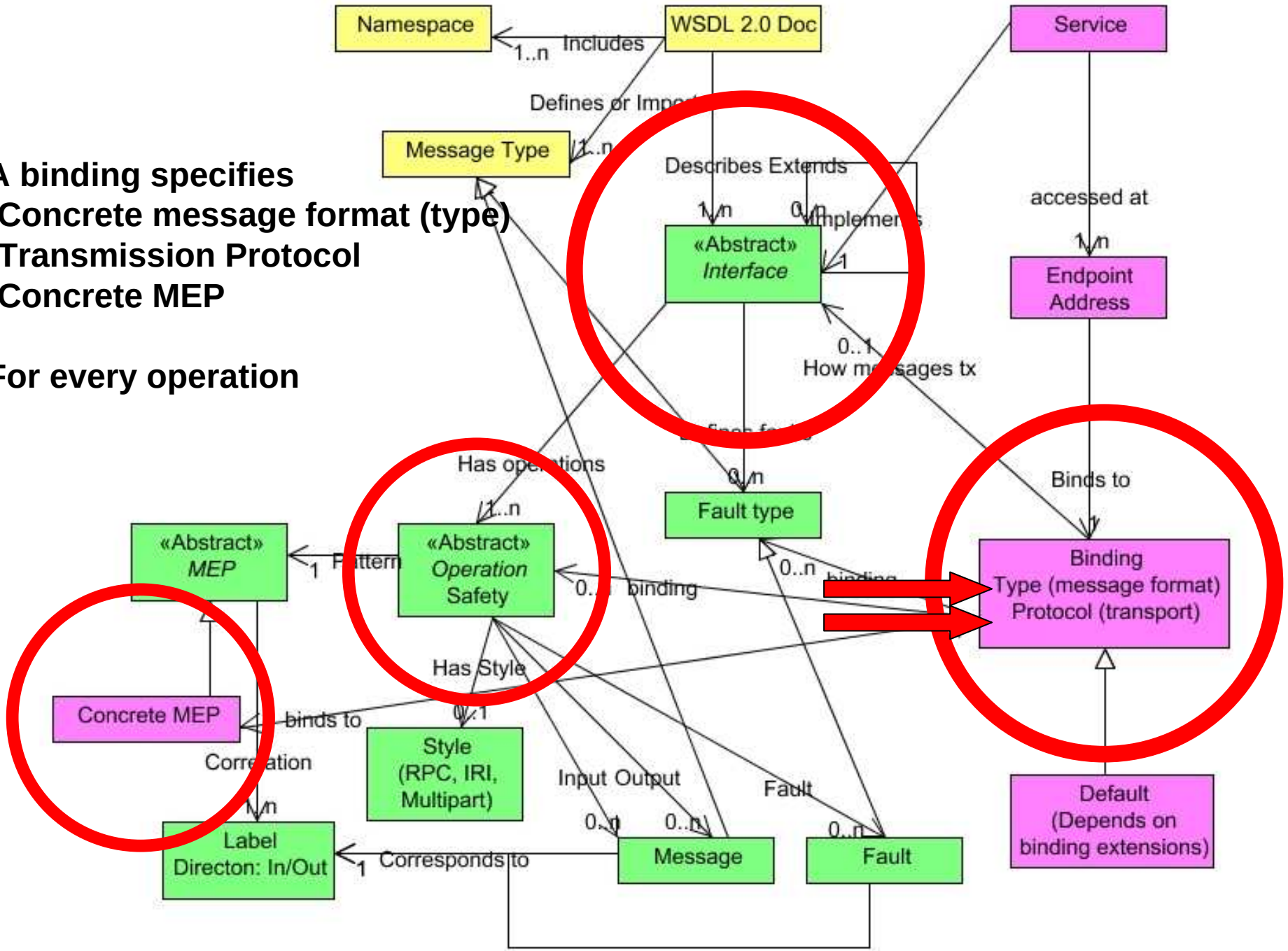


A binding specifies
-Concrete message format (type)
-Transmission Protocol
-Concrete MEP



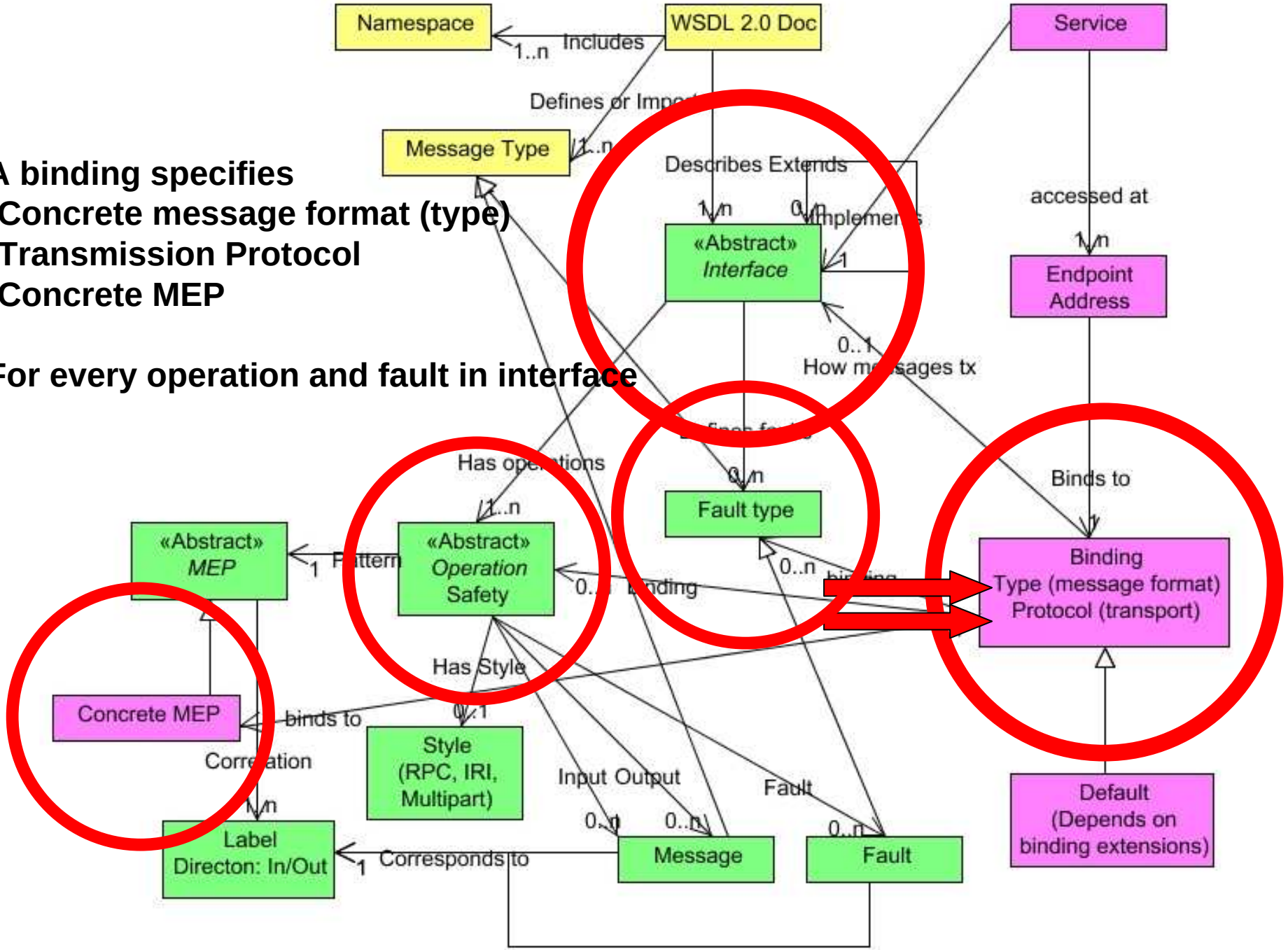
A binding specifies
-Concrete message format (type)
-Transmission Protocol
-Concrete MEP

For every operation

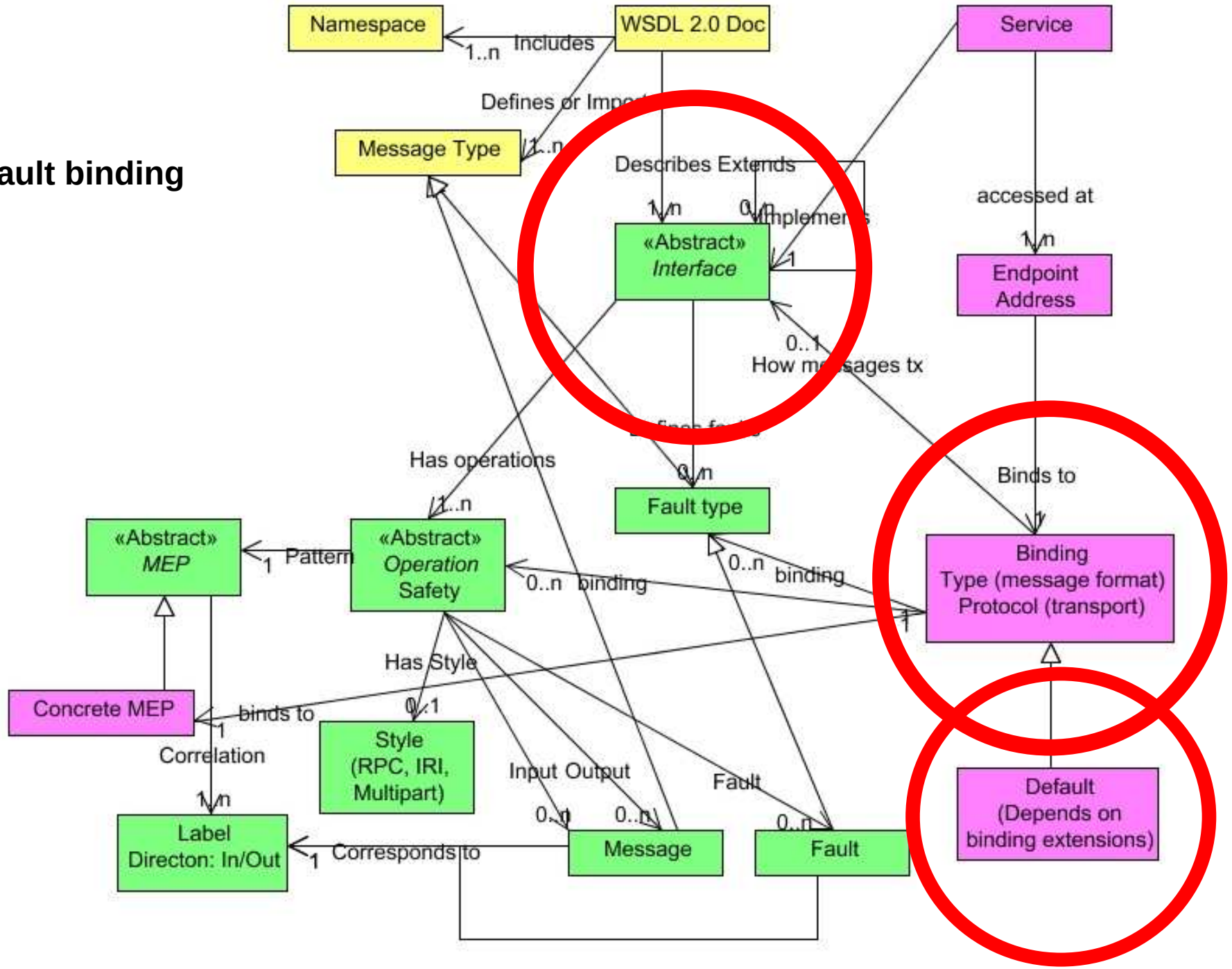


A binding specifies
-Concrete message format (type)
-Transmission Protocol
-Concrete MEP

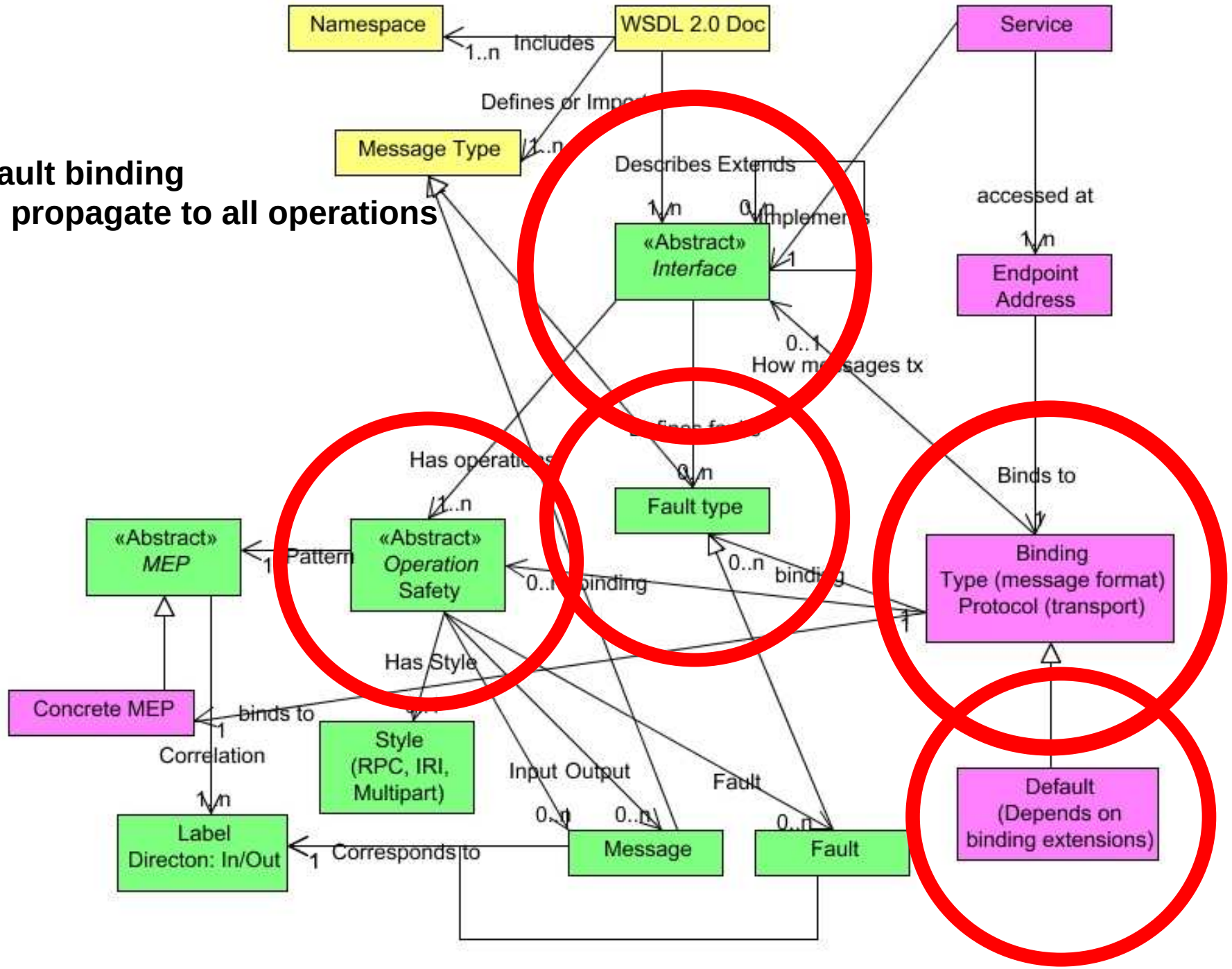
For every operation and fault in interface



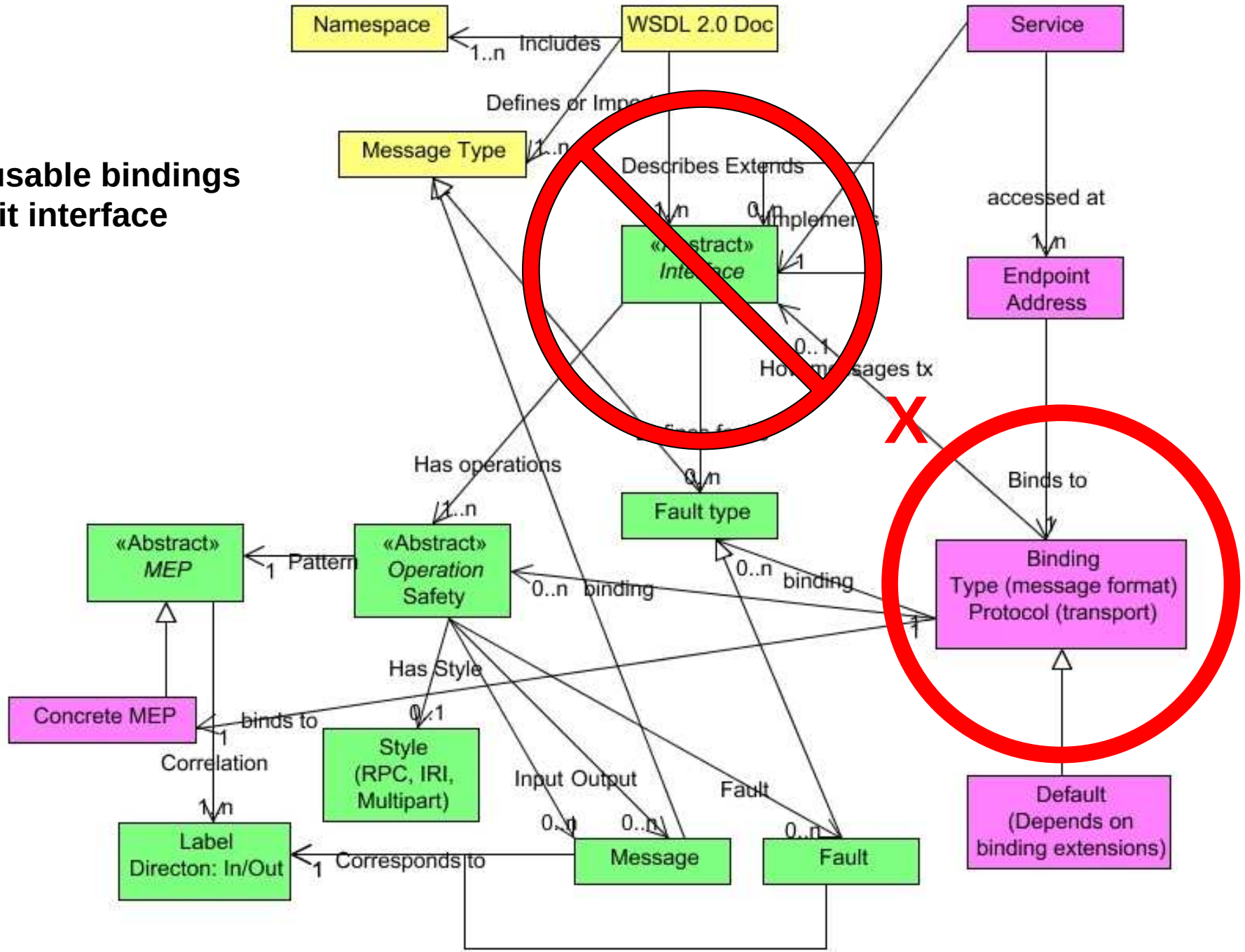
Default binding



Default binding
Will propagate to all operations

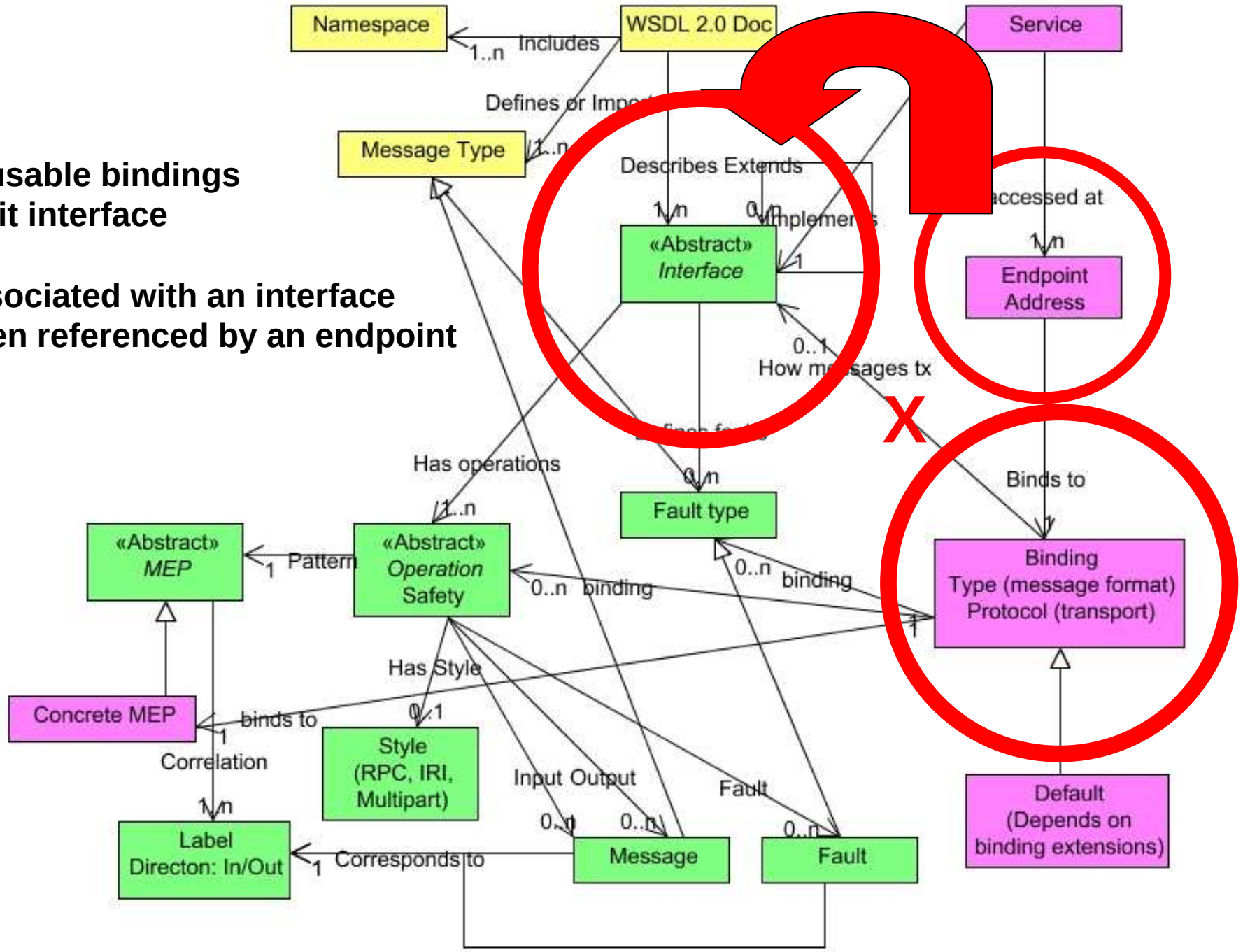


Reusable bindings
Omit interface



Reusable bindings
Omit interface

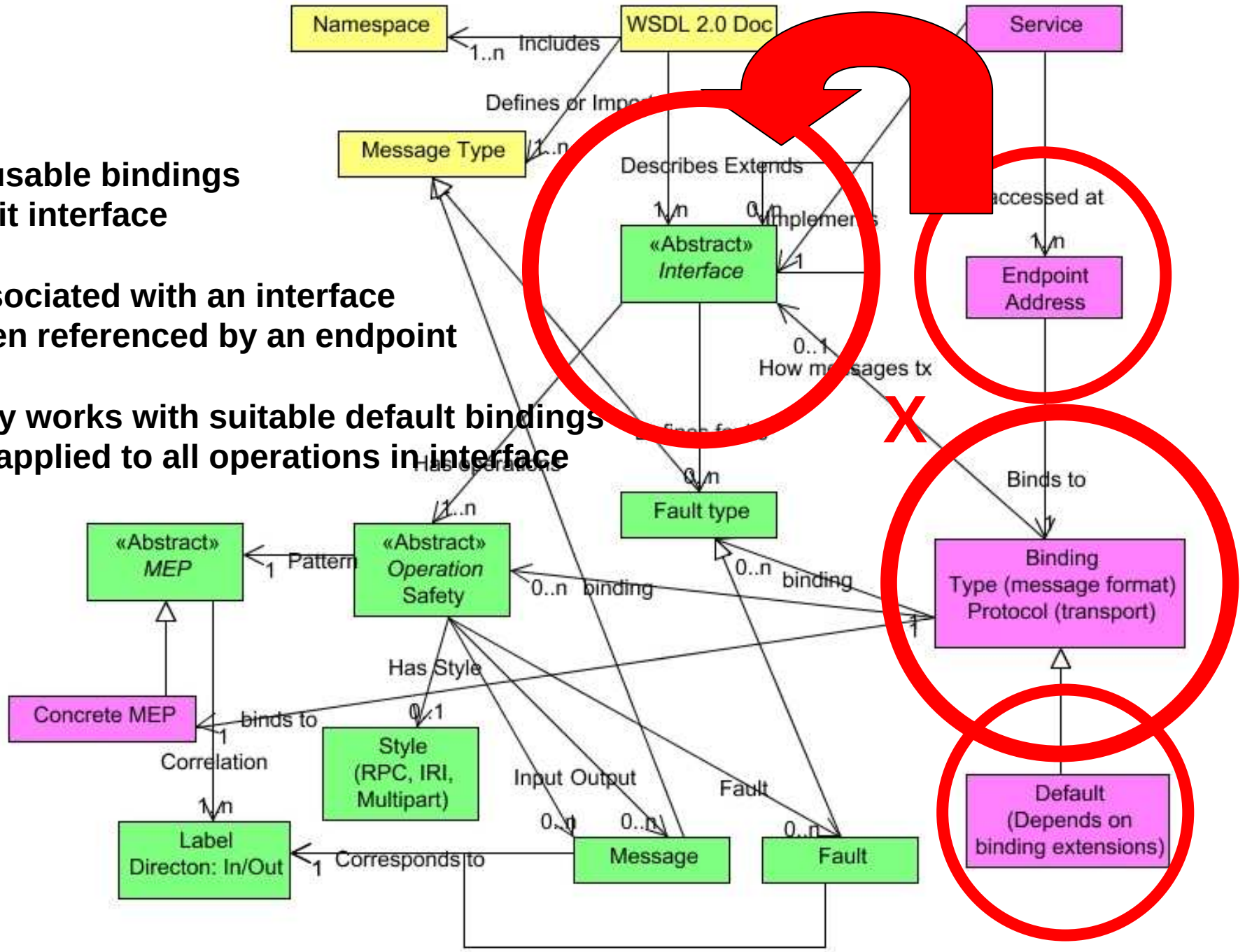
Associated with an interface
when referenced by an endpoint



Reusable bindings
Omit interface

Associated with an interface
when referenced by an endpoint

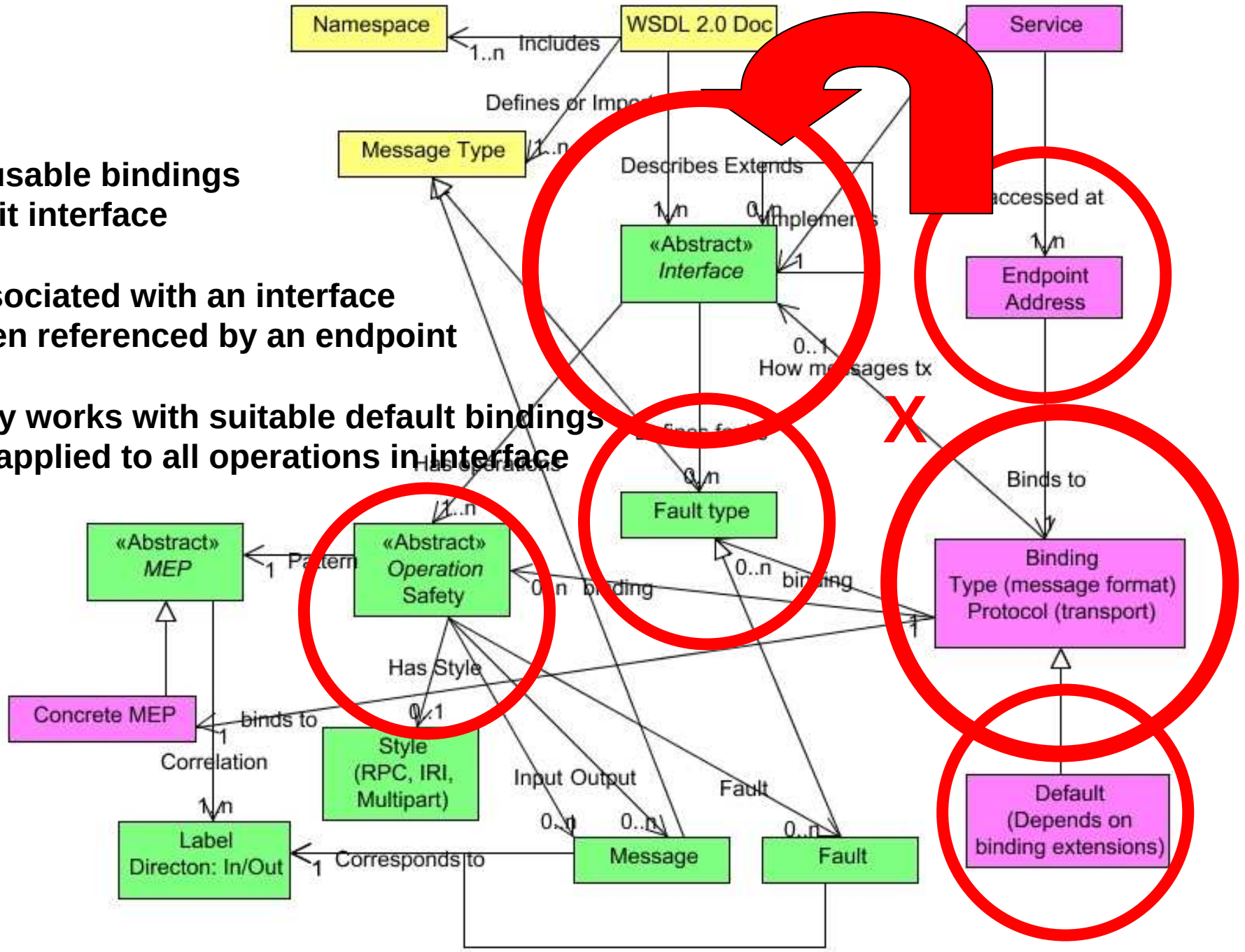
Only works with suitable default bindings
As applied to all operations in interface



Reusable bindings
Omit interface

Associated with an interface
when referenced by an endpoint

Only works with suitable default bindings
As applied to all operations in interface



Example of non-reusable binding

```
<binding name="reservationSOAPBinding"
  interface="tns:reservationInterface"
  type="http://www.w3.org/2006/01/wsdl/soap"
  wsoap:protocol="http://www.w3.org/2003/05/so
  ap/b
  <ope                                    ty"
     wsoap:mep="http://www.w3.org/2003/05/soap
  /mep/soap-response"/>
  <fault ref="tns:invalidDataFault"
     wsoap:code="soap:Sender"/>
</binding>
```

The interface whose message format and transmission protocols we are specifying.

Example of non-reusable binding

```
<binding name="reservationSOAPBinding"
  interface="tns:reservationInterface"
  type="http://www.w3.org/2006/01/wsdl/soap"
  wsoap:protocol="http://www.w3.org/2003/05/so
  ap/bindings/HTTP">
  <o                                    ility"
                                        03/05/soap
  /r
  <fault ref="tns:invalidDataFault"
     wsoap:code="soap:Sender"/>
</binding>
```

Kind of concrete message format to use, in this case SOAP 1.2.

Example of non-reusable binding

```
<binding name="reservationSOAPBinding"
  interface="tns:reservationInterface"
  type="http://www.w3.org/2006/01/wsdl/soap"
  wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP">
  <operation ref="tns:opCheckAvailability"
     wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>
  <fault ref="tns:invalidDataFault"
     wsoap:code="soap:Sender"/>
</binding>
```

Specifies the underlying transmission protocol that should be used, in this case HTTP.

Example of non-reusable binding

```
<binding name="reservationSOAPBinding"
  interface="tns:reservationInterface"
  type="http://www.w3.org/2006/01/wsdl/soap"
  wsoap:protocol="http://www.w3.org/2003/05/so
  ap/bindings/HTTP">
  <operation ref="tns:opCheckAvailability"
  wsoap:mep="http://www.w3.org/2003/05/soap/me
  p/soap-response"/>
  <fault r
     wsoap
</binding>
```

referencing the previously defined opCheckAvailability operation in order to specify binding details for it.

Example of non-reusable binding

```
<binding name="reservationSOAPBinding"
  interface="tns:reservationInterface"
  type="http://www.w3.org/2006/01/wsdl/soap"
  wsoap:protocol="http://www.w3.org/2003/05/so
  ap/bindings/HTTP">
  <operation ref="tns:opCheckAvailability"
  wsoap:mep="http://www.w3.org/2003/05/soap/me
  p/soap-response"/>
  <fault ref="
     wsoap:cod
</binding>
```

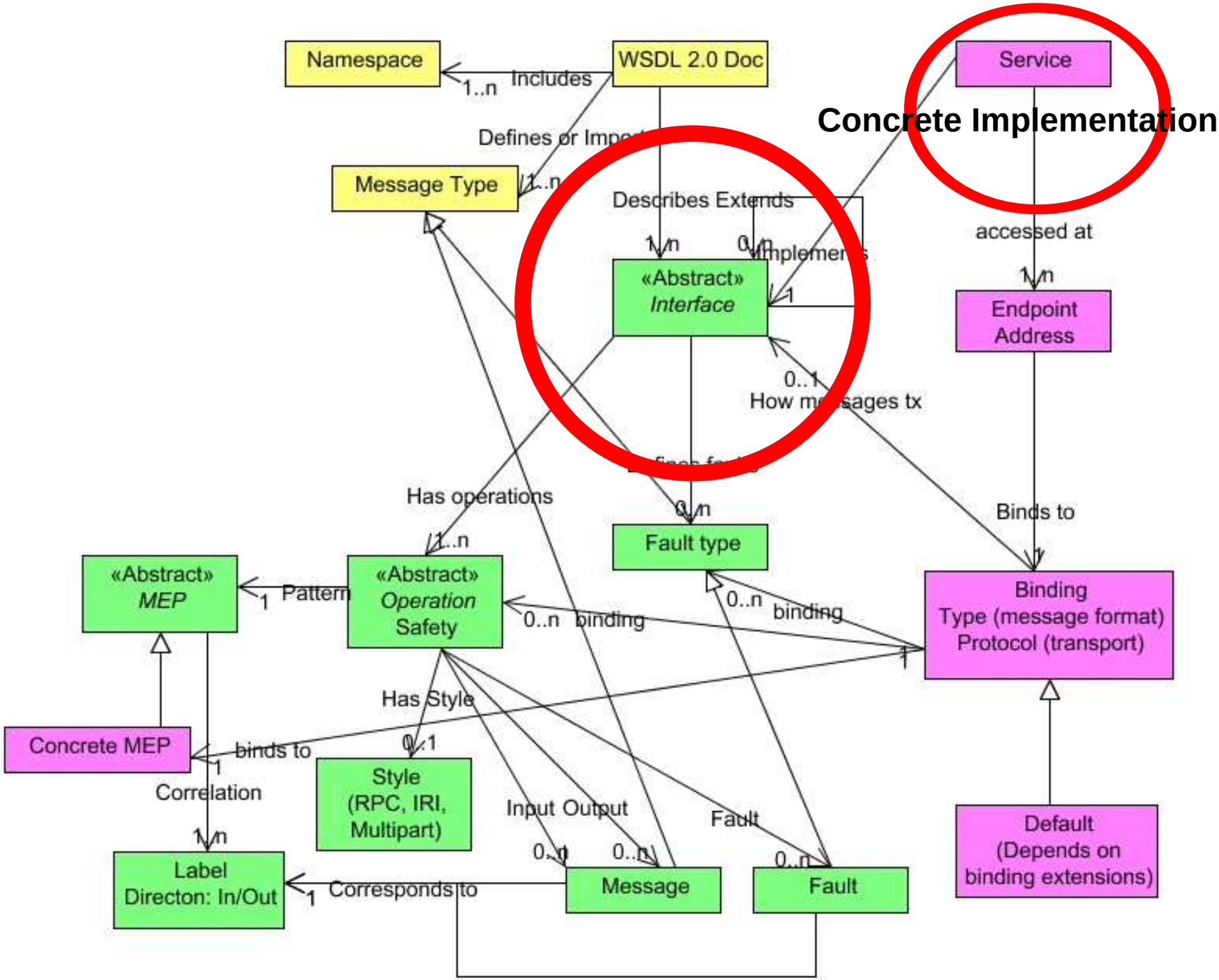
The SOAP MEP that will be used to implement the abstract WSDL 2.0 MEP (in-out) that was specified when the opCheckAvailability operation was defined.

Reusable Bindings

- Reusable bindings
  - A binding can either be reusable (applicable to any interface) or non-reusable (specified for a particular interface).
  - A binding that omits an interface is reusable.
  - The binding becomes associated with a particular interface when it is referenced from an endpoint (because an endpoint is part of a service, and the service specifies a particular interface that it implements).
  - Since a reusable binding does not specify an interface, reusable bindings cannot specify operation-specific details.
  - Therefore, reusable bindings can only be defined using binding extensions that have suitable defaulting rules, such that the binding information only needs to be explicitly supplied at the interface level.
- What are endpoints?

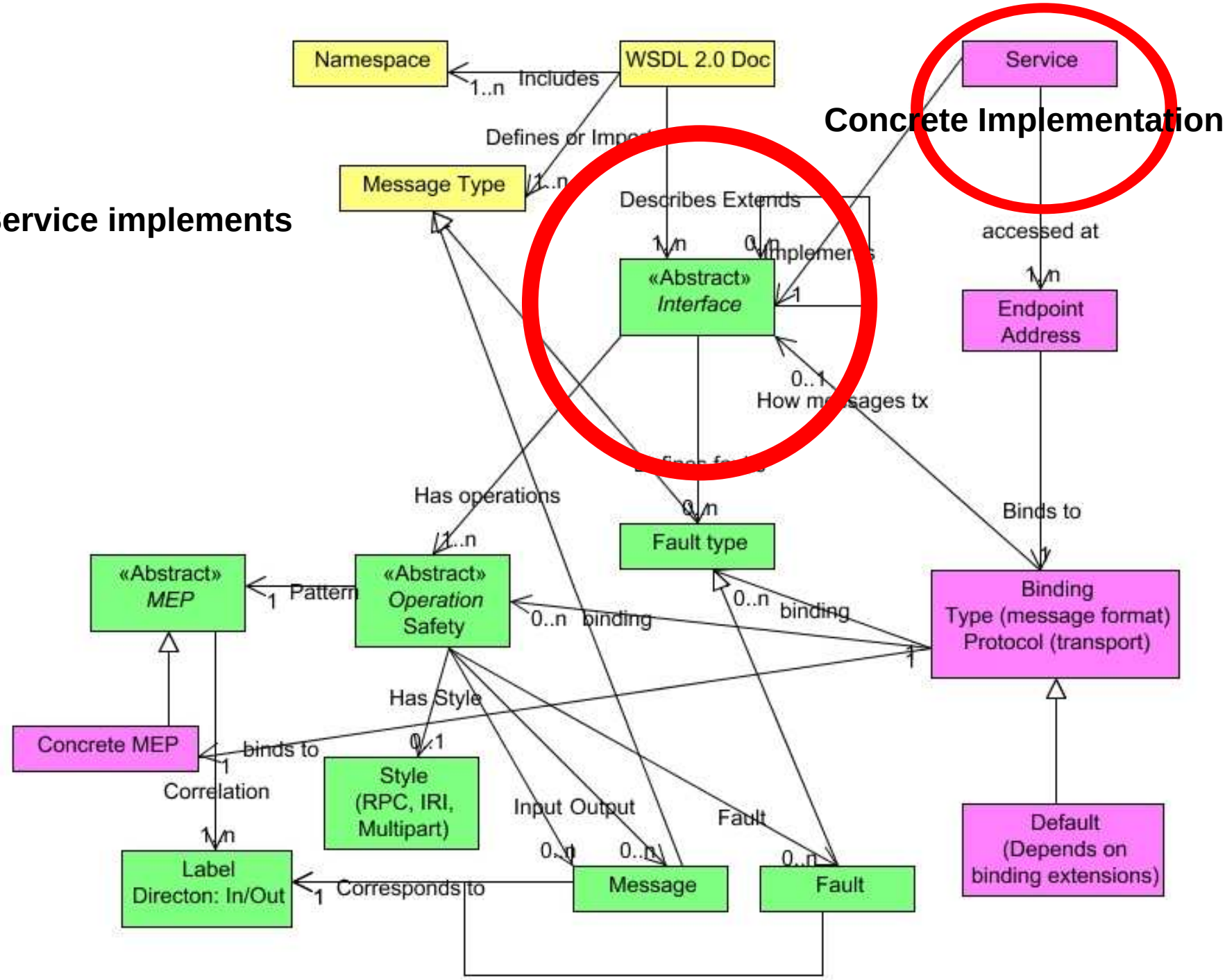
Service

- A c*oncrete implementation* of an interface
- Bindings specify *how* messages will be transmitted
  - Services specify *where* the service can be accessed
- A WSDL 2.0 *service* specifies
  - a single interface that the service will support
  - and a list of *endpoint* locations where that service can be accessed
    - Each endpoint must reference a binding
      - to indicate what protocols and transmission formats are to be used at that endpoint.
  - *A service is only permitted to have one interface.*

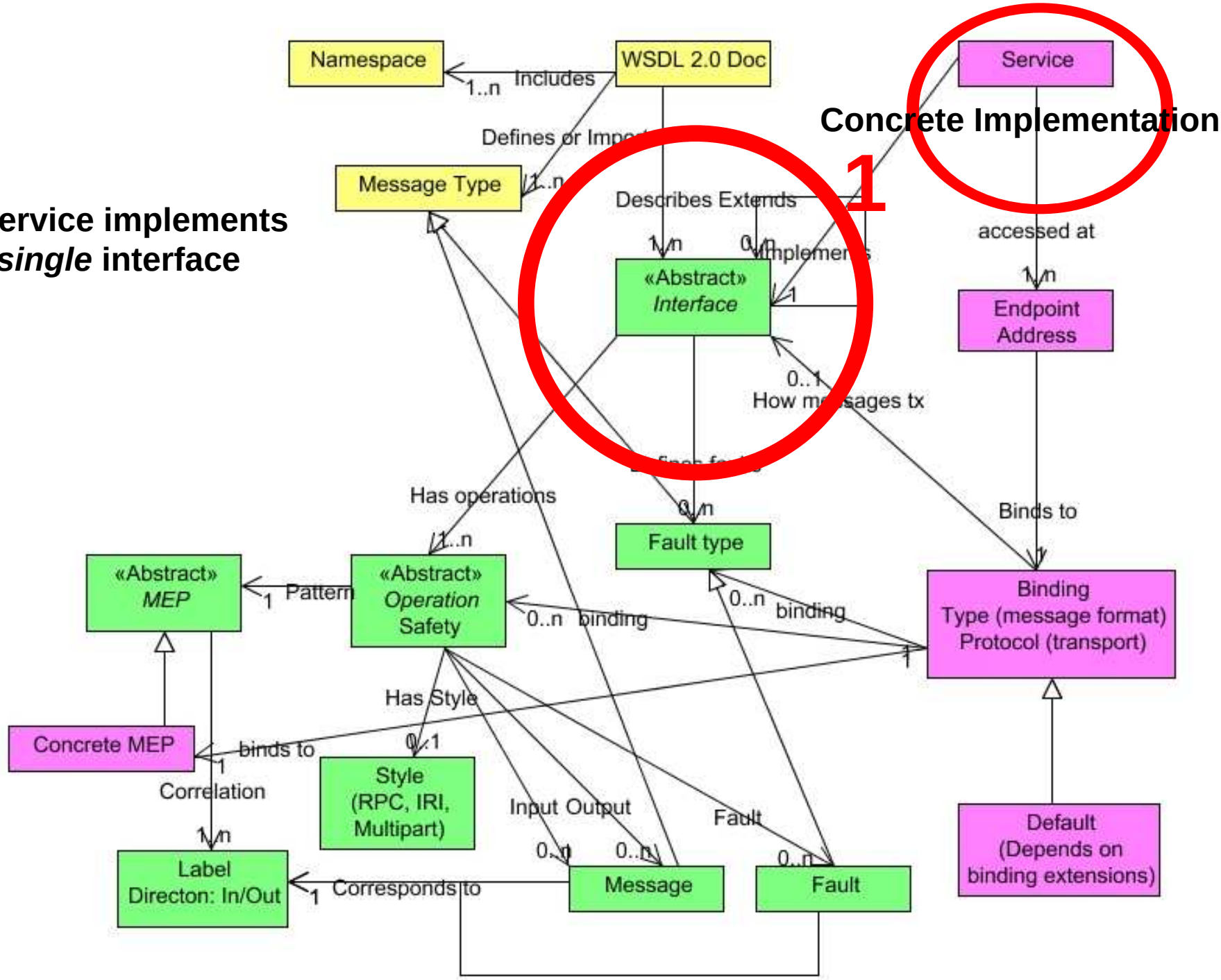

Concrete Implementation
Namespace
WSDL 2.0 Doc
Includes
1..n
Service
Defines or Imports
Message Type
1..n
Describes Extends
1..n
0..n
Implements
«Abstract»
Interface
1
accessed at
1..n
Endpoint
Address
0..1
How messages tx
Has operations
1..n
0..n
Fault type
Binds to
«Abstract»
MEP
1
Pattern
«Abstract»
Operation
Safety
0..n
binding
0..n
binding
1
Binding
Type (message format)
Protocol (transport)
Has Style
Concrete MEP
binds to
1
0..1
Style
(RPC, IRI,
Multipart)
Correlation
1..n
Input Output
Fault
0..n
0..n
0..n
Label
Directon: In/Out
1
Corresponds to
Message
Fault
Default
(Depends on
binding extensions)

A Service implements

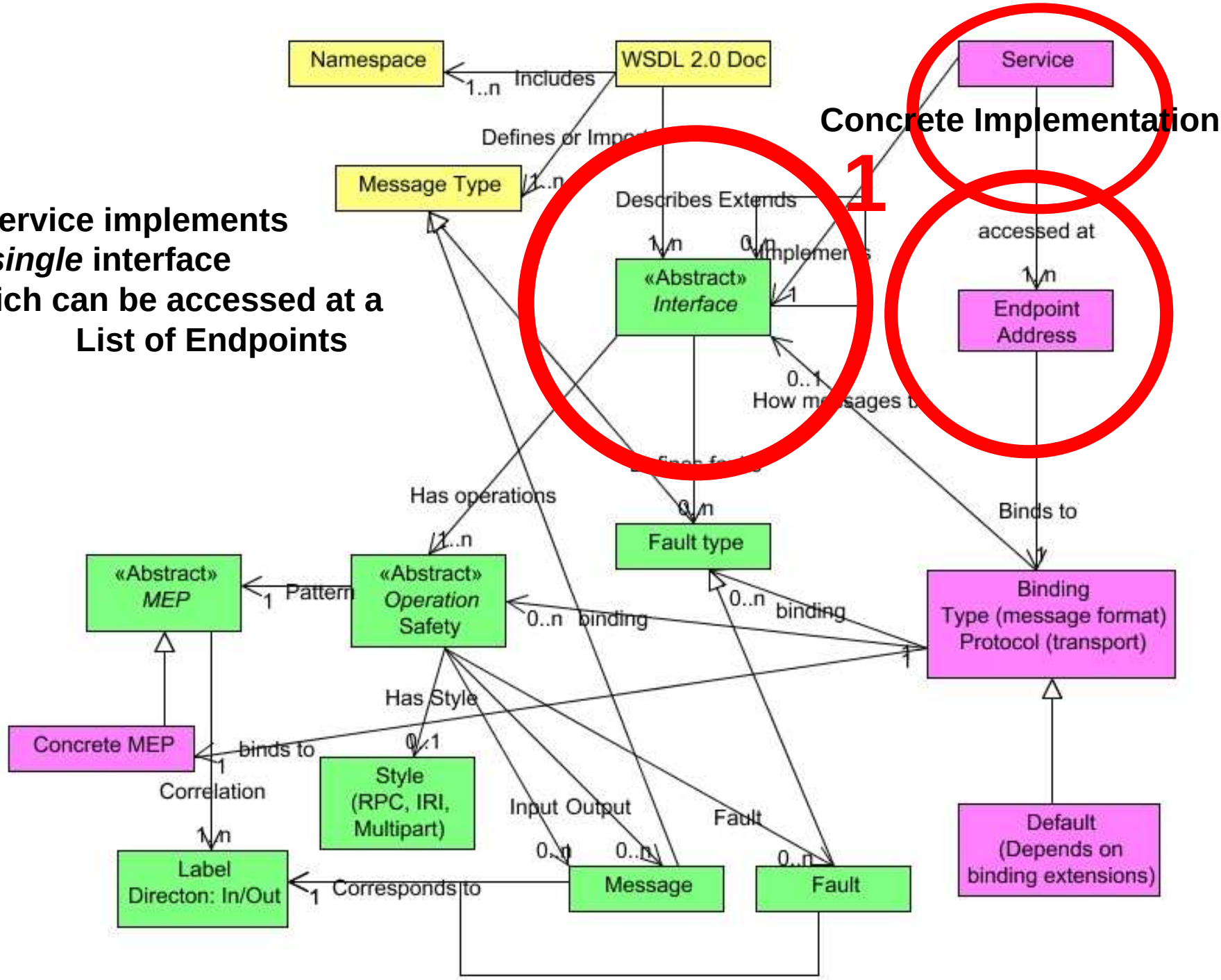
Concrete Implementation



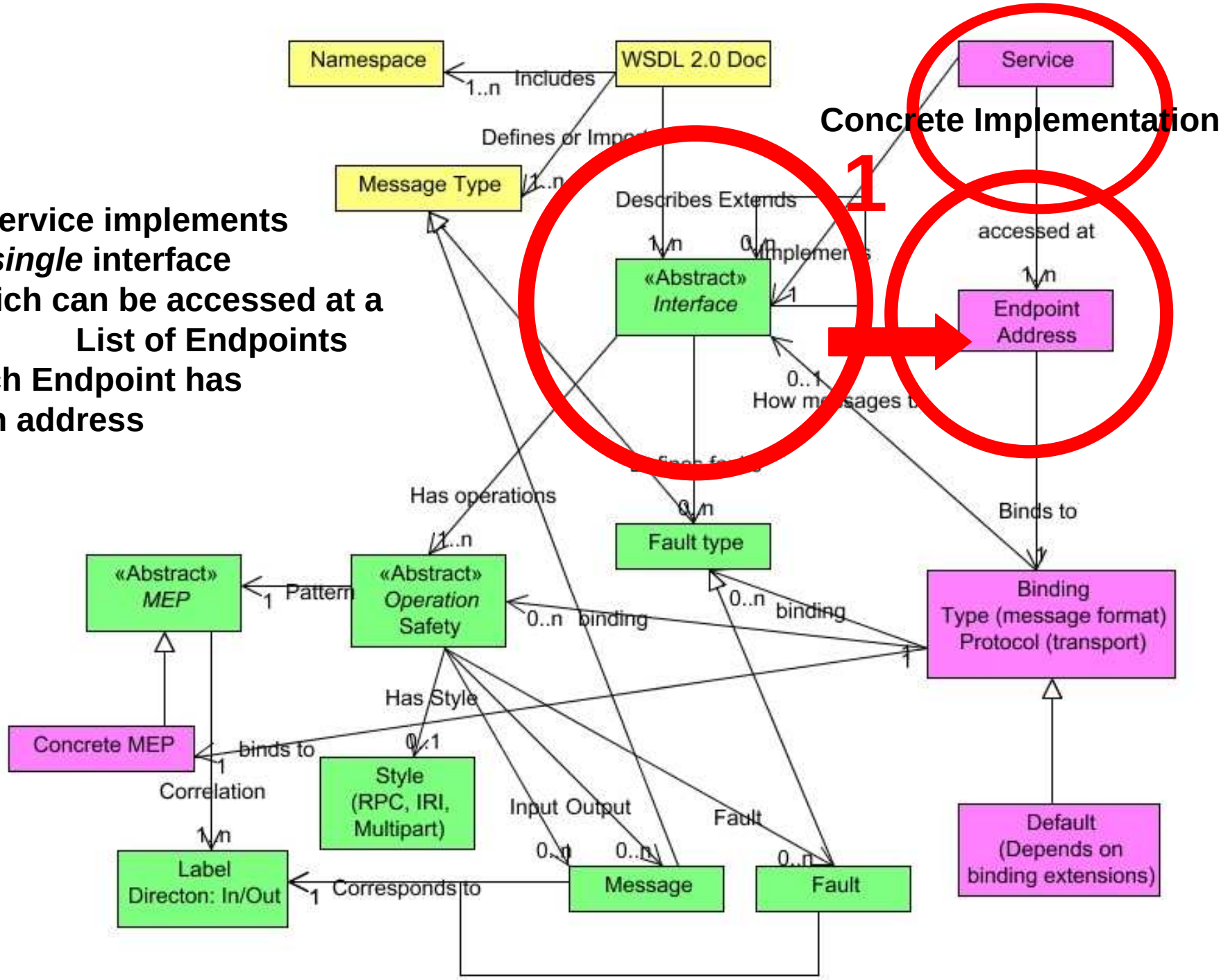
A Service implements
- A *single* interface



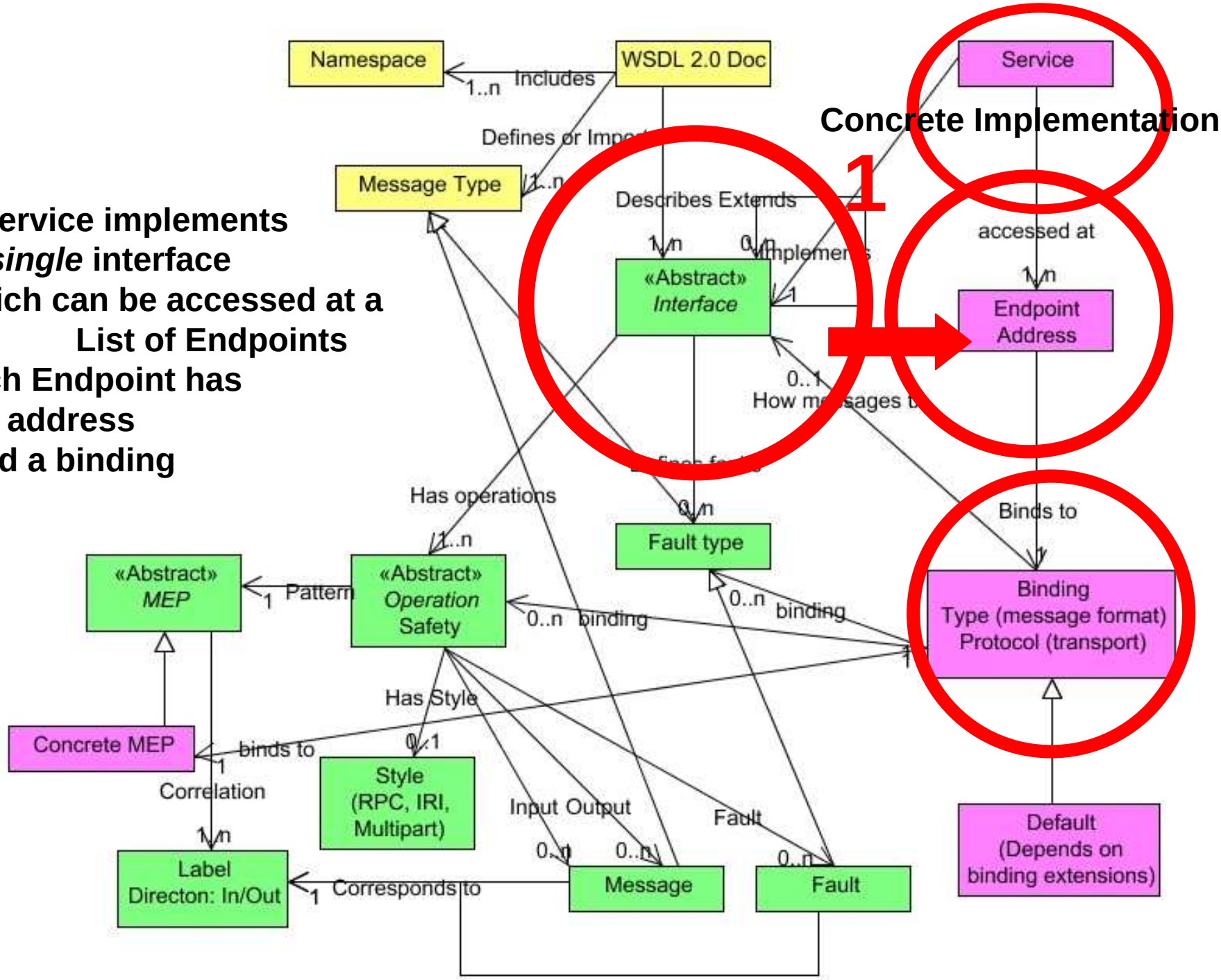
A Service implements
-A *single* interface
Which can be accessed at a
List of Endpoints

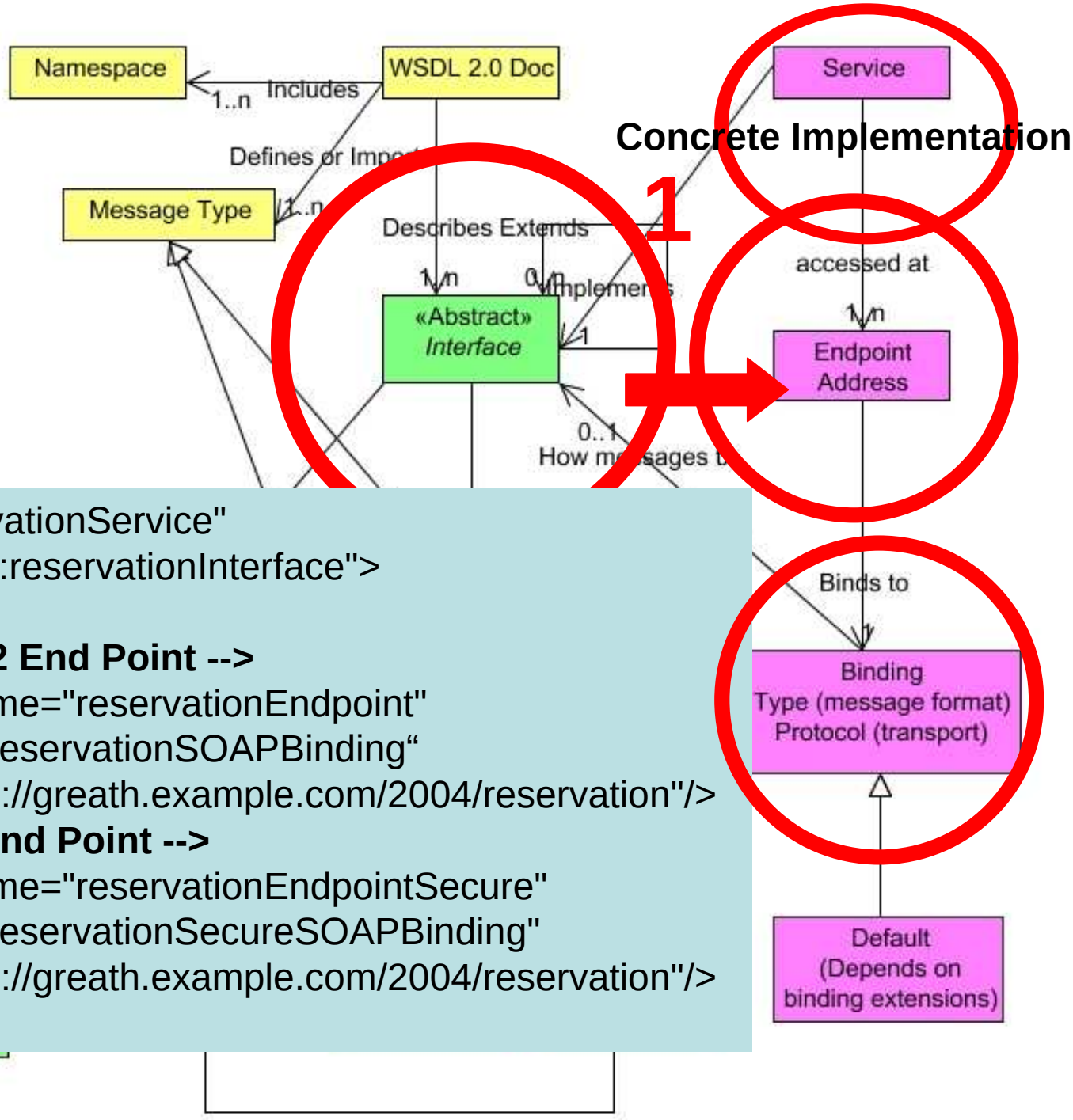


A Service implements
-A *single* interface
Which can be accessed at a
List of Endpoints
Each Endpoint has
- An address



A Service implements
-A *single* interface
Which can be accessed at a
List of Endpoints
Each Endpoint has
-An address
-And a binding






```
<service name="reservationService"
        interface="tns:reservationInterface">

        <!-- SOAP 1.2 End Point -->
        <endpoint name="reservationEndpoint"
        binding="tns:reservationSOAPBinding“
        address="http://greath.example.com/2004/reservation"/>
        <!-- Secure End Point -->
        <endpoint name="reservationEndpointSecure"
        binding="tns:reservationSecureSOAPBinding"
        address="http://greath.example.com/2004/reservation"/>
</service>
```

New MEPs – E.g. Subscription with challenge

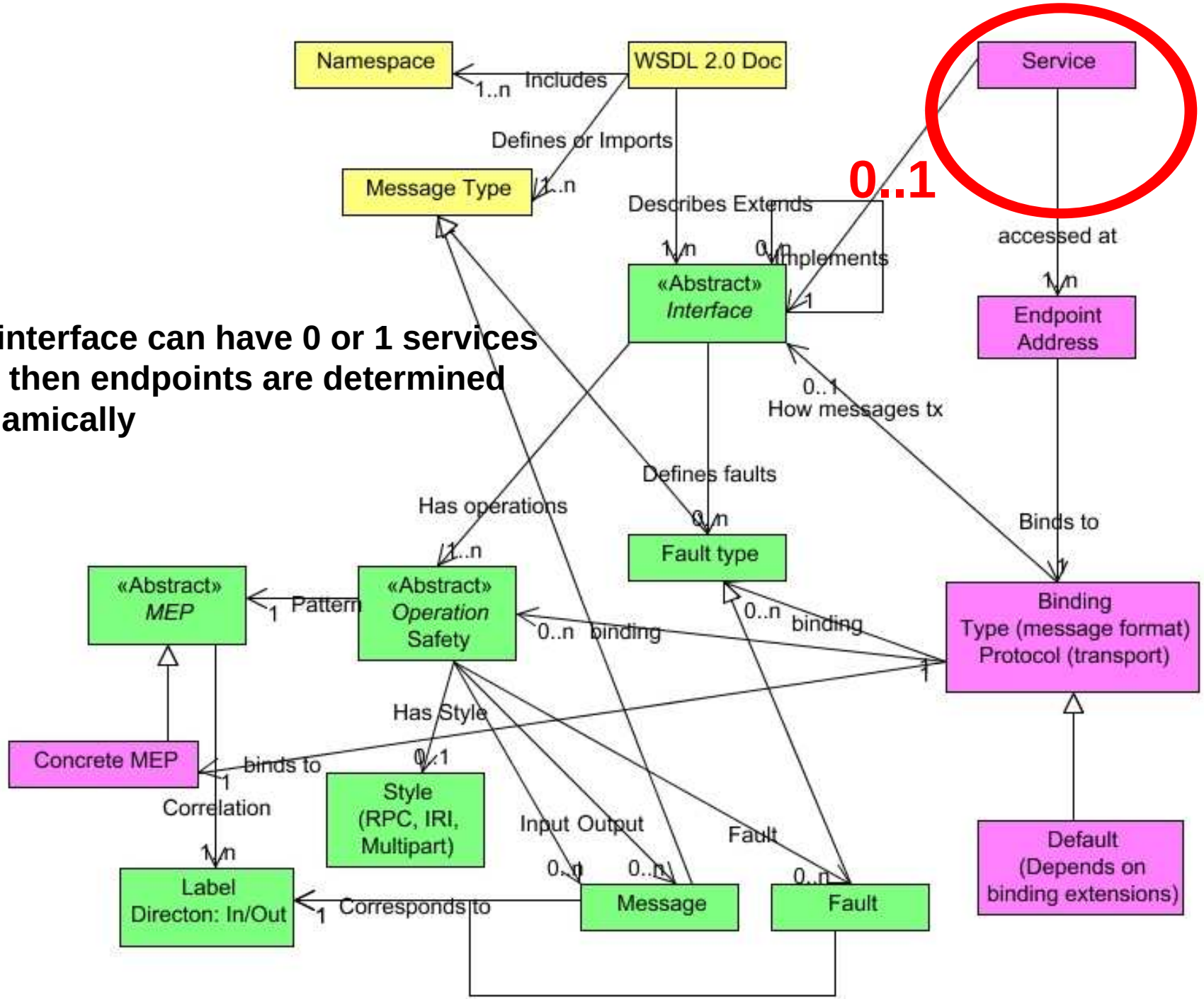
```
<interface name=“subscriptionInterface">
<operation name="opSubscribe"
   pattern="http://www.example.com/webservices/meps/confirm
   ed-challenge">
   <input messageLabel="Request" element=". . ." />
   <output messageLabel="Challenge" element=". . ." />
   <input messageLabel="Confirmation" element=". . ." />
   <output messageLabel="Response" element=". . ." />

   </operation> <operation name="opUnsubscribe"
   pattern="http://www.example.com/webservices/meps/confirm
   ed-challenge">
   …
   </operation> . . .
</interface>
```

Web Service “Hyperlinking”

- Services should be first-order objects
  - C.f. Web Hyperlinks
  - A reference to a service
- Motivating examples
  - Grid Factory Service
    - Creates new service instance and returns address to client
      - May be on a different machine to the factory.
  - Reservation service
    - Creates a reservation for a customer, and returns the address of the reservation/customer service instance:
      - Client uses this address to query, change, and confirm reservation.
      - No need for further authentication etc

An interface can have 0 or 1 services
If 0, then endpoints are determined dynamically



Example – reservation “cart”

- Say you’ve requested a room at the hotel via the reservation service
  - at the address http://greath.example.com/2004/reservation
- Get a reservation *endpoint* back which includes the address
  - http://greath.example.com/2004/reservation/AX43PG98CD42
- Can now access the service *at this address* to
  - retrieve your reservation details (no input message required), or update it, cancel it, etc.

The reservationDetails interface – no service!

```
<interface name="reservationDetailsInterface">
    <operation name="retrieve“
    pattern="http://www.w3.org/2006/01/wsdl/in-out">
    <input messageLabel="In" element="#none" />
    <output messageLabel="Out" element="wdetails:reservationDetails" />
    </operation>

    <operation name="update“
    pattern="http://www.w3.org/2006/01/wsdl/in-out">
    <input messageLabel="In" element="wdetails:reservationDetails" />
    <output messageLabel="Out" element="wdetails:reservationDetails" />
    </operation>
</interface>

<binding name="reservationDetailsSOAPBinding"
    interface="tns:reservationDetailsInterface" …
 </binding>
```

Endpoints?

```
<simpleType name="reservationDetailsSOAPEndpointType"
        wsdlx:binding="wdetails:reservationDetailsSOAPBinding“>
        <restriction base="anyURI"/>
</simpleType>
```

- The schema defines the simple type reservationDetailsSOAPEndpointType which is based on xs:anyURI and has the annotation wsdlx:binding = "wdetails:reservationDetailsSOAPBinding" which means that the URI is the address of a Reservation Details Web service endpoint that implements the wdetails:reservationDetailsSOAPBinding binding.
  - The wsdli:wsdlLocation attribute is used to define the location of the WSDL 2.0 document that defines the wdetails:reservationDetailsSOAPBinding binding.
  - This annotated simple type is used to define the reservationDetailsSOAPEndpoint element which will be used in the Reservation List service.
- The endpoints returned by the reservation service; E.g.
  - <details:reservationDetailsSOAPEndpoint> http://greath.example.com/2004/reservation/HSG635 </details:reservationDetailsSOAPEndpoint>
  - allow the client to call the service with knowledge of binding.

Discussion

- Is this just the OGSA/GT3 grid handle mechanism? (service instances returned by a factory service)
  - How does GT4 model service instances?
- How does this relate to MEPs and (stateful) service correlation?

Message correlation (and stateful service modelling)

- WS-Addressing specification already provides a disambiguation mechanism.
  - It defines a required [action] property whose value is always present in a message delivery.
  - The value of the action property can be used to disambiguate the message by the receiver and there is a well defined way to associate actions to messages in WS-Addressing specifications.
  - WS-Addressing also provides an appropriate default action value that identifies each message uniquely.
- How can WS-Addressing be used in WSDL 2.0?
  - To achieve OGSA-like stateful service interactions?

WSRF and WS-Addressing

- WSRF builds on WS-addressing specification to achieve the same goals (references to stateful services)
  - First, it adopts the endpoint reference construct defined in the WS-addressing specification as an XML syntax for identifying Web service endpoints.
    - It then defines a particular usage pattern for endpoint references, the *implied resource pattern*, in which the *reference properties* field of the endpoint reference contains an identifier of a specific WS-resource associated with the Web service.
    - These two pieces of information are the logical equivalent of the addressing content of the OGSI defined GSR.
  - Second, rather than distinguishing between two fixed types of names, immutable GSHs and potentially mutable GSRs, it introduces (in WS-RenewableReferences) a mechanism for associating with any endpoint reference (not just one that refers to a WS-resource) a “resolver service.
    - Specifically, WS-RenewableReferences allows a renewable reference policy to be associated with an endpoint reference.
    - This WS-policy statement can include an assertion concerning the ReferenceResolver for obtaining a new reference for a particular service.

Features and Properties

- A feature can describe a non-functional requirement (which may be required or not) for most WSDL elements:
  - E.g. security, transactions
  - Abstract and concrete
- Features can have properties
  - E.g. authentication, authorisation, using different “strength” algorithms.
  - Can describe constraints on values.
- Similar to Annotations/Aspects?
  - and therefore easily supported by containers with support for annotations for deployment/configuration?

Abstract feature

```
<interface name="reservationInterface">
   <operation name="makeReservation">
   <feature
   uri="http://features.example.com/2005/sec
   urityFeature" required="true"/> …
</operation>
</interface>
```

Operations “considered bad”?

- Are web service *operations* “bad”? Hang over from distributed Objects?
  - Coupling a web service implementation to its interface by directly implementing the operations – maintenance nightmare.
  - Why should client have to tell service “how” to process a message (by calling a specific operation)?
- Why expose object “methods” to client?
  - Simplify the MEPs?
    - one simple MEP per operation
  - Implicit semantics in the operation name?
    - e.g. methodAdd1ToInteger(Int)!
- Better to “read” the interface as a whole, not as operation by operation description?
  - I.e. If you send these types of messages to the service you get these types back (depending on form and content of messages)
  - Essentially treating services as single operation/polymorphic messaging systems.
  - In general this would require a far more sophisticated MEP specification, as the conversation with the service would need to be modelled as a rich process (e.g. BPEL).
    - E.g. A reservation service with a single “process” operation would need specify what message types, order, number, exceptions, results are permitted and produced.

“Semantics”?

- WSDL-S
  - Attempts to add semantics (functionality) to WSDL 2.0.
  - Pre and post conditions/effects for operations.
    - Operation-centric?
    - Could semantics be defined over message exchanges?
- Semantic Annotations for WSDL W3C Working Group - SAWSDL
  - new W3C spec
  - Semantic annotations for any WSDL element

SOA practice: Service Modelling, Service Development

- XML isn’t very human writable/readable
- Use tools, MDA, and intelligent containers:
  - For interface modelling and definition
  - For implementation development
  - For client-side
  - For deployment/hosting
    - E.g. Enterprise Service Bus (E.g. MULE) or IoC container (E.g. Spring) to isolate implementation from interface dependencies

Model-driven service development?

- WS can be developed using UML Models
  - Component model generates types, interfaces
  - Interaction diagrams for MEPs.
  - Stereotypes and PSM deal with binding details
  - E.g. Will Provost (WSDL 1.1)
    - Could be extended to WSDL 2.0.

The End

Confused?

"Silly old Bear," he said, "what were you doing? First you went round twice by yourself, and then Piglet ran after you and you went round again together, and then you were just going round a fourth time"

"I have been Foolish and Deluded," said he, "and I am a Bear of No Brain at All."

"Anyhow," he said, "it is nearly Lunch Time."
(Well, … not yet)

Thanks to Tony Rogers (Computer Associates) for his WSDL 2.0 talk at ANU/CSIRO in June 2006. Any mistakes in this talk are all mine.

Further information: http://www.w3.org/2002/ws/desc/, http://www.w3.org/TR/wsdl20-primer/

This is a longer version of the talk presented at AusWebSIG 06.