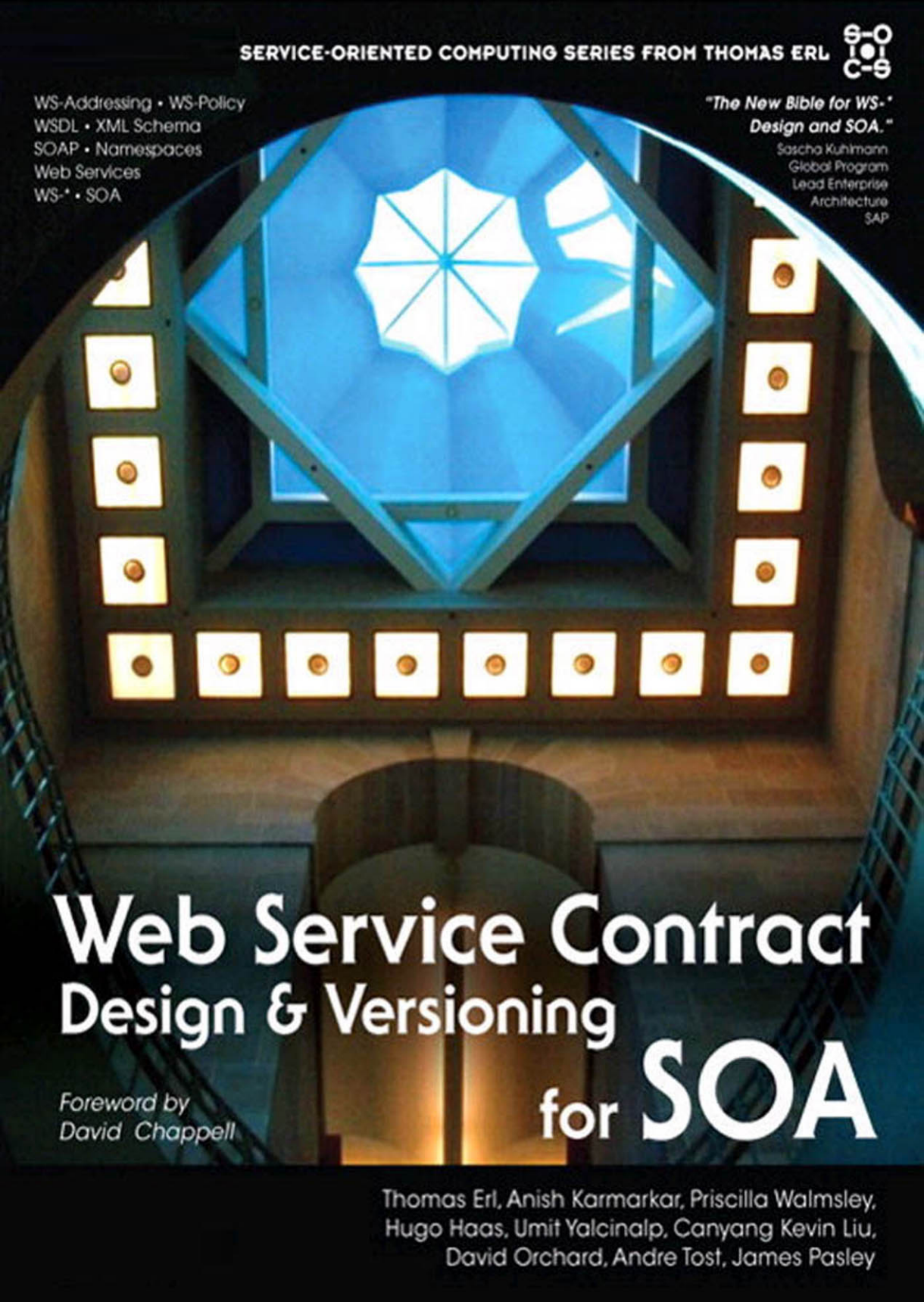
SERVICE-ORIENTED COMPUTING SERIES FROM THOMAS ERL
S-O
I-O
C-S
WS-Addressing • WS-Policy
WSDL • XML Schema
SOAP • Namespaces
Web Services
WS-* • SOA
"The New Bible for WS-* Design and SOA."
Sascha Kuhlmann
Global Program
Lead Enterprise
Architecture
SAP
Web Service Contract
Design & Versioning
for SOA
Foreword by
David Chappell
Thomas Erl, Anish Karmarkar, Priscilla Walmsley,
Hugo Haas, Umit Yalcinalp, Canyang Kevin Liu,
David Orchard, Andre Tost, James Pasley

Praise for this Book

"This compendium is the new bible for WS-* design and SOA. The structured approach allows beginners, experts and executives to understand and execute successfully on an SOA Strategy. Thomas Erl and his team did once again a great job."

—Sascha Kuhlmann
Global Program Lead Enterprise Architecture, SAP

"A first-rate and in-depth dive into service contract and interface design that elaborates on the best book in the field, ***SOA Principles of Service Design***. Over the years Thomas's books have provided a foundation for defining our SOA standards (for over 200 services) and contain the latest material and case studies which comprise the singular most important advice for those implementing large-scale SOAs."

—Bob Hathaway
Senior SOA Architect
SOA Object Systems, Starwood Hotels

"Anyone tasked with publishing and evolving Web services 'contract first' will welcome this book, packed full of authoritative explanations and advice from authors who truly understand their subject."

—Paul Downey
Chief Web Services Architect, BT

"This book is an excellent resource for anybody trying to get a comprehensive understanding of the technical foundation of Web services based service-oriented architecture. It is written by the experts who have worked on the development of the underlying world-wide standards."

—Michael Bechauf
Vice President of Standards Strategy and Developer Programs, SAP AG

```
    <soap12:operation soapAction="" soapActionRequired=""
      style="" wsdl:required="" />
    <input>
      <soap12:body parts="" use="" encodingStyle=""
        namespace="" wsdl:required="" />
      <soap12:header message="" part="" use=""
        encodingStyle="" namespace="" wsdl:required="">
      <soap12:headerfault message=""
        part="" use="" encodingStyle="" namespace=""
        wsdl:required="" />
      </soap12:header>
    </input>
    <output>
      <!-- same as input -->
    </output>
    <fault>
      <soap12:fault name="" use="" encodingStyle=""
        namespace="" wsdl:required="" />
    </fault>
  </operation>
</binding>
<service ...>
  <port ...>
    <soap12:address location="" wsdl:required="" />
  </port>
</service>
```

Example 8.12
A `binding` construct highlighting SOAP 1.2 extensibility elements.

It's clear that a SOAP 1.2 `binding` construct mimics the SOAP 1.1 `binding` construct with only a few changes. Differences between SOAP 1.1 and 1.2 constructs are highlighted in the previous example and further explained here:

- *`soap12`: Prefix* – The SOAP 1.2 binding constructs are defined in the new namespace `http://schemas.xmlsoap.org/wsdl/soap12/`, as represented by the `soap12:` prefix in the previous example. This new URI can be used in binding definitions to indicate that an interface is bound to the SOAP 1.2 protocol.
- *`soapActionRequired` Attribute* – This optional attribute of the binding `soap12:operation` element provides a means of indicating whether a `soapAction` value is required.

NOTE

The XML schema for the SOAP 1.2 binding extension can be found at http://schemas.xmlsoap.org/wsdl/soap12/wsdl11soap12.xsd.

The `wsdl:required` Attribute

The global `wsdl:required` attribute defined in the WSDL1.1 specification can be used with any extension element to indicate whether the extension is required or not. Although it is leveraged by SOAP 1.2 extension elements, this attribute is technically not new with SOAP 1.2 binding because it can also be used with SOAP 1.1 extensions (or any other extensions).

CASE STUDY EXAMPLE

Given that Steve has already created a `binding` construct with SOAP 1.1 extensibility elements, adjusting it to support SOAP 1.2 is pretty straightforward. The differences from Example 8.11 are highlighted in the following example:

```
<binding name="bdPO-SOAP12HTTP" type="ptPurchaseOrder">
  <soap12:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="opSubmitOrder">
    <soap12:operation soapAction=
      "http://actioncon.com/submitOrder/request"
      soapActionRequired="true" wsdl:required="true"/>
    <input>
      <soap12:body use="literal"/>
    </input>
    <output>
      <soap12:body use="literal"/>
    </output>
  </operation>
  <operation name="opCheckOrderStatus">
    <soap12:operation soapAction=
      "http://actioncon.com/checkOrder/request"
      soapActionRequired="true" wsdl:required="true"/>
    <input>
      <soap12:body use="literal"/>
    </input>
    <output>
      <soap12:body use="literal"/>
    </output>
  </operation>
  <operation name="opChangeOrder">
```

```
    <soap12:operation soapAction=
      "http://actioncon.com/changeOrder/request"
      soapActionRequired="true" wsdl:required="true"/>
    <input>
      <soap12:body use="literal"/>
    </input>
    <output>
      <soap12:body use="literal"/>
    </output>
  </operation>
  <operation name="opCancelOrder">
    <soap12:operation soapAction=
      "http://actioncon.com/cancelOrder/request"
      soapActionRequired="true" wsdl:required="true"/>
    <input>
      <soap12:body use="literal"/>
    </input>
    <output>
      <soap12:body use="literal"/>
    </output>
  </operation>
</binding>
```

Example 8.13
A `binding` construct with SOAP 1.2 extensibility elements.

SUMMARY OF KEY POINTS

- The `binding` construct is comprised of WSDL language elements and extensibility elements from other languages that are used to bind the contract to communication technologies.
- Within the hierarchy of a `binding` construct, the settings established by extensibility elements can be inherited by child elements unless child elements are assigned their own extensibility elements.
- An important consideration with the binding definition overall is combination of the settings for the `use` and `style` attributes.

8.3 Service and Port Definitions

Probably the simplest of all primary WSDL elements are the `service` and `endpoint` constructs. Their role, of course, is essential for a Web service to function, but their syntax is straightforward, as it focuses on establishing one or more addresses.

The `service` and `port` Elements

After a binding definition is completed, we need to assign it a physical network address that consumer programs will use to locate and invoke the Web service. The WSDL 1.1 elements used to establish this last part of the concrete description are `service` and `port`.

Here's what they look like:

```
<service name="svPurchaseOrder">
  <port name="purchaseOrder-http-soap11"
    binding="tns:bdPO-SOAP11HTTP">
    <soap11:address location=
      "http://actioncon.com/services/soap11/purchaseOrder"/>
  </port>
</service>
```

Example 8.14
A parent `service` construct with a `port` element.

The `port` element defines a physical location (often called an *endpoint*) for a binding definition. It identifies the binding definition via the required `binding` attribute. Note that a `port` can only reference one `binding` element which, in turn, can only reference one `portType` element. This means that a port essentially represents the deployed implementation of one single port type (interface).

The actual `port` element address value is defined via an extensibility element because each transport may have a different format for expressing the address. This extensibility element is located within the `port` construct, as follows:

```
<port name="purchaseOrder" binding="tns:bdPurchaseOrder">
  <soap11:address location=
    "http://actioncon.com/services/soap11/purchaseOrder"/>
</port>
```

Example 8.15
A sample `port` construct with a SOAP 1.1 `address` extensibility element.

A `port` construct cannot contain more than one address. If you have a set of related `port` elements, you can use the `service` construct to group them together.

NOTE

The naming of the `service` element can be confusing. The contents of this element by no means defines a Web service as a whole. You can think of the `port` element details more as invocation information required to create a service instance.

WS-I Guideline

When a `service` construct has multiple `port` elements, how are these ports supposed to be related to each other? Can the `port` elements implement different `portType` constructs? The WSDL 1.1 specification doesn't provide clear answers for these questions.

WSDL 2.0

The `port` element is renamed to `endpoint` in the WSDL 2.0 specification.

The WS-I Basic Profile (and WSDL 2.0) states that when a `service` construct hosts several `port` elements, that these ports should be related to the same `portType` (when the port type employs multiple binding definitions). So a best practice advocated by the WS-I is to not group `port` elements associated with different `portType` definitions into one `service` construct.

CASE STUDY EXAMPLE

In the previous examples, we've discussed how Steve at ActionCon wants to design the Purchase Order service to support both SOAP versions 1.1 and 1.2. To accomplish this, he needs to group the two `port` elements into a `service` construct, as follows:

```
<service name="svPurchaseOrder">
  <port name="purchaseOrder-http-soap11"
    binding="tns:bdPO-SOAP11HTTP">
    <soap11:address location=
      "http://actioncon.com/services/soap11/purchaseOrder"/>
  </port>
  <port name="purchaseOrder-http-soap12"
    binding="tns:bdPO-SOAP12HTTP">
    <soap12:address location=
      "http://actioncon.com/services/soap12/purchaseOrder"/>
  </port>
</service>
```

Example 8.16

A `service` construct with two `port` elements providing support for SOAP 1.1 and 1.2.

Just for fun, let's assume that Steve decides to implement this service with bindings for SOAP 1.1 over HTTP *and* SMTP. In this case, the `service` construct would resemble something like this:

```
<service name="svPurchaseOrder">
  <port name="purchaseOrder-http" binding="tns:bdPO-SOAP11HTTP">
    <soap11:address location=
      "http://actioncon.com/services/purchaseOrder"/>
  </port>
  <port name="purchaseOrder-smtp"
    binding="tns:bdPurchaseOrder-smtp">
    <soap11:address location=
      "mailto:purchaseOrderProcessing@actioncon.com"/>
  </port>
</service>
```

Example 8.17
A `service` construct with two `port` elements providing support for SOAP 1.1 via HTTP and SMTP.

SUMMARY OF KEY POINTS

- The elements used to define a concrete `binding` construct are `operation`, `input`, `output`, and `fault`.
- Binding extensibility elements for SOAP 1.1 are defined as part of the WSDL 1.1 specification, but not for SOAP 1.2.
- The WSDL 1.1 elements used to establish the physical location for a binding definition are `service` and `port`.

8.4 A Complete WSDL Definition (Including the Concrete Description)

In order to provide an end-to-end representation of a WSDL definition, the following example shows both completed abstract and concrete descriptions together.