

# The SOA Magazine

## Feature Article



### Understanding WS-Policy Part II: Operator Composition Rules and Attachments

by Umit Yalcinalp and Thomas Erl

Published: March 23, 2009 (SOA Magazine Issue XXVII: March 2009)

[Download PDF](#)

[Digg This](#) • [De.licio.us](#) • [Slashdot](#) • [Technorati](#) • [StumbleUpon](#) • [Google Bookmark](#)

*Abstract: The following article is an excerpt from the new book "Web Service Contract Design and Versioning for SOA" [REF-1] Copyright Prentice Hall/Pearson PTR and SOA Systems Inc. Note that some chapter references were intentionally left in the article, as per requirements from Prentice Hall.*

### Introduction

The WS-Policy language provides a set of features that allow for the definition of complex and sophisticated policies and that also enables you to choose from a range of options as to how policies can be attached and associated with WSDL definitions. In this, the second of a two-part article about WS-Policy, we demonstrate the use of operators and related composition rules. We then move onto an exploration of policy attachment methods, including external and embedded attachment options...

### Operator Composition Rules

The rules of operator composition are often referred to as "operator algebra," but don't let that term give you flashbacks to high school math class. This section is really just a more involved discussion about the inherent properties of operator elements and how these properties can affect the structures you build.

It's important to understand these effects because you can combine operators in multiple ways to create highly complex policy expressions. In fact, one of the more interesting aspects of operator compositions is that you can produce composite policies comprised of policies created by yourself and others.

The WS-Policy defined operators `wsp:All` and `wsp:ExactlyOne` have the following properties that form the basis of operator composition rules:

- idempotent
- commutative
- associative
- `wsp:All` distributes over `wsp:ExactlyOne`

The following sections look at how these rules are applied in practice and further discuss the use of empty operator elements and how `wsp:Policy` compares to `wsp:All`.

### Idempotent Rule

Idempotency means that nesting multiple occurrences of operators within each other is equivalent to a single occurrence. In other words, applying the same operator to itself multiple times yields the same result.

This means that the following operator structure...

```
<wsp:All>
  <wsp:All>
    <wsp:All>
      <wsam:Addressing><wsp:Policy/></wsam:Addressing>
      <wsrmp:RMAssertion><wsp:Policy/></wsrmp:RMAssertion>
    </wsp:All>
  </wsp:All>
</wsp:All>
```

...is equivalent to:

```
<wsp:All>
  <wsam:Addressing><wsp:Policy/></wsam:Addressing>
  <wsrmp:RMAssertion><wsp:Policy/></wsrmp:RMAssertion>
</wsp:All>
```

### Commutative Rule

Commutativity means that the ordering of the children within the operator construct is insignificant.

In other words, the following operator structure...i

```
<wsp:All>
  <wsam:Addressing><wsp:Policy/></wsam:Addressing>
  <wsrmp:RMAssertion><wsp:Policy/></wsrmp:RMAssertion>
</wsp:All>
```

...is equivalent to:

```
<wsp:All>
  <wsrmp:RMAssertion><wsp:Policy/></wsrmp:RMAssertion>
  <wsam:Addressing><wsp:Policy/></wsam:Addressing>
</wsp:All>
```

### Associative Rule

This rule enables the streamlining of operator constructs by allowing for the removal of unnecessary nesting.

This means that the following operator structure...

```
<wsp:All>
  <wsam:Addressing><wsp:Policy/></wsam:Addressing>
  <wsp:All>
    <wsrmp:RMAssertion><wsp:Policy/></wsrmp:RMAssertion>
    <argt:responseGuarantee/>
  </wsp:All>
</wsp:All>
```

...is equivalent to:

```
<wsp:All>
  <wsam:Addressing><wsp:Policy/></wsam:Addressing>
  <wsrmp:RMAssertion><wsp:Policy/></wsrmp:RMAssertion>
  <argt:responseGuarantee/>
</wsp:All>
```

**wsp:All** is Distributive Over **wsp:ExactlyOne**

This rule is primarily used for the normalization of policy expressions. It is very useful when constructing a set of alternatives at the top layers of an expression with further alternatives nested within the children. By applying the distributive property of `wsp:All`, we have obtained an equivalent expression with a distinct alternative that is expressed at the root.

What this means is that the following operator structure...

```
<wsp:All>
  <wsp:ExactlyOne>
    <wsam:Addressing><wsp:Policy/></wsam:Addressing>
    <wsrmp:RMAssertion><wsp:Policy/></wsrmp:RMAssertion>
  </wsp:ExactlyOne>
</wsp:All>
```

...is equivalent to:

```
<wsp:ExactlyOne>
  <wsp:All>
    <wsam:Addressing><wsp:Policy/></wsam:Addressing>
  </wsp:All>
  <wsp:All>
    <wsrmp:RMAssertion><wsp:Policy/></wsrmp:RMAssertion>
  </wsp:All>
</wsp:ExactlyOne>
```

Let's take a more complex example to illustrate distribution. The following operator structure...

```
<wsp:All>
  <wsp:ExactlyOne>
    <wsam:Addressing><wsp:Policy/></wsam:Addressing>
    <wsrmp:RMAssertion><wsp:Policy/></wsrmp:RMAssertion>
  </wsp:ExactlyOne>
  <wsp:ExactlyOne>
    <argt:responseGuarantee/>
  </wsp:ExactlyOne>
</wsp:All>
```

...is equivalent to:

```
<wsp:ExactlyOne>
  <wsp:All>
    <wsam:Addressing><wsp:Policy/></wsam:Addressing>
    <argt:responseGuarantee/>
  </wsp:All>
  <wsp:All>
    <wsrmp:RMAssertion><wsp:Policy/></wsrmp:RMAssertion>
    <argt:responseGuarantee/>
  </wsp:All>
</wsp:ExactlyOne>
```

In the first example, each `wsp:ExactlyOne` construct establishes a policy alternative but the second has only one assertion. Because they are further wrapped in a `wsp:All` construct, the actual alternatives of the policy expression are:

Alternative #1: `wsam:Addressing` + `argt:responseGuarantee`

Alternative #2: `wsrmp:RMAssertion` + `argt:responseGuarantee`

This is more clearly conveyed in the second example due to the restructuring of the `wsp:ExactlyOne` and `wsp:All` elements.

We explore distributive operator concepts further in Chapter 16.

## Empty Operators

Sometimes policy operators do not contain policy assertions. In this case, they are expressed as follows:

- indicates a policy with zero assertions.
- indicates a policy with zero policy alternatives.

Knowing this syntax is relevant because empty content can affect the meaning of an operator composition as follows:

```
<wsp:All>
  <wsp:ExactlyOne>
    <wsam:Addressing><wsp:Policy/></wsam:Addressing>
    <wsrmp:RMAssertion><wsp:Policy/></wsrmp:RMAssertion>
  </wsp:ExactlyOne>
  <wsp:ExactlyOne/>
</wsp:All>
```

The policy expression in this example is essentially equivalent to a policy with zero policy alternatives because the empty operator statement (highlighted) has no assertions.

## Equivalence

So far, we've been studying the `wsp:All` and `wsp:ExactlyOne` operator elements. However, we have also seen the `wsp:Policy` element used with various child elements as well. You may be wondering how the `wsp:Policy` element fits into the composition rules.

A `wsp:Policy` element as an operator is equivalent to the `wsp:All` element. Therefore, an empty policy is equivalent to , a policy that contains zero assertions.

## Attaching Policies to WSDL Definitions

WS-Policy as a framework by itself would not be very useful without the ability to associate policy expressions to something. In this section we explore how policy expressions can be attached to different parts of a WSDL definition by using features from the WS-PolicyAttachment standard.

There are two ways of attaching policies to WSDL documents:

- The policy expression code can be embedded within the WSDL definition and then utilized via native references that can be attached directly to WSDL elements as child extensibility elements.
- Policy expressions can reside externally in a separate WS-Policy definition document, which is then referenced within the WSDL document via an external attachment mechanism.

The *Attaching Policies to WSDL Definitions* section in this chapter and the *Reusability and Policy Centralization* section in Chapter 16 explain each of these options. But first, we need to cover a few basic attachment-related topics in the next two sections.

## Policy Attachment Points and Policy Subjects

Whatever part of a WSDL definition we attach a policy to is referred to as a *policy attachment point*. For example, we could have a policy attached to an `operation` element and then another to one of that operation's `message` elements.

Within WSDL 1.1, the following elements represent common policy attachment points:

- `service` element
- `port` element
- `binding` element
- `portType` element
- `operation` element
- `message` element

The policy attachment specification organizes these policy attachment points into the following four distinct *policy subjects*:

- Service
- Endpoint
- Operation
- Message

Each policy subject represents a pre-defined scope within the overall WSDL definition. For example, one expression may apply to a particular message only, whereas another might have a scope that corresponds to an entire operation. In the latter case, the policy expression would be applicable to all message definitions that fall within that operation.

Let's discuss the policy subjects individually.

#### *Service Policy Subject*

This subject corresponds to the `service` element within the concrete description of a WSDL document. Therefore, a policy expression attached to the service subject will apply to all messages associated with the `port` constructs that fall within the `service` construct. This makes this subject the parent or master subject in that a policy expression attached at this level pretty much affects all message definitions within the WSDL document.

We haven't yet covered how policies get attached to parts of the WSDL definition, but at this stage, let's at least show the insertion point for the policy code so that we have a better idea as to how a policy can be attached to the service policy subject:

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

#### *Endpoint Policy Subject*

The service policy subject seemed pretty straightforward, but things get a little more complicated with this one. The term "endpoint" in this case refers to the combination of the following three elements:

- `port`
- `binding`
- `portType`

So if you attach a policy expression to any one of these three elements, you have attached it to the endpoint policy subject.

To better understand this, first think about how these three elements are related in terms of messaging. A port type definition encompasses operations that have message definitions. But the port type needs to be bound to a concrete protocol via a corresponding binding definition which, in turn, needs an address provided by the port definition. In other words, all three elements are related to the same set of messages (those originally defined in the `portType` construct).

Because, as we stated earlier, policy expressions are all about establishing qualities and properties associated with messaging, grouping these three elements together as one policy subject makes a lot of sense.

Here's a look at where the policy code would go in the case of a binding element:

```
<binding name="bdPO-SOAP12HTTP" type="ptPurchaseOrder">
  <!-- Insert Policy Expression Here -->
  <soap12bind:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="opSubmitOrder">
    ...
  </operation>
  ...
</binding>
```

### *Operation Policy Subject*

You might have noticed that as we work through this list, the application scope of a policy gets smaller. Now we're moving down the ladder to the operation level.

Following the same rationale we just explained for the endpoint policy subject, you attach a policy to an operation policy subject when you attach it to either of the following two elements:

- the `operation` element of a `portType` construct
- the `operation` element of a `binding` construct

Again, this is because these two elements are so closely related and both represent the same set of message definitions.

Let's take a look at where the policy expression goes:

```
<operation name="opSubmitOrder">
  <!-- Insert Policy Expression Here -->
  <soap12bind:operation
    soapAction="http://action.com/submitOrder/request"
    soapActionRequired="true" required="true"/>
  <input>
    <soap12bind:body use="literal"/>
  </input>
  <output>
    <soap12bind:body use="literal"/>
  </output>
</operation>
```

### *Message Policy Subject*

This last policy subject simply allows you to pinpoint a message definition with a policy expression. All of the preceding subjects grouped messages together, but there will be times when an expression is so specific that it only applies to a particular message.

Because a given message will be defined via message elements in the abstract and concrete descriptions, this subject represents any one of the following possible element pairs:

- the input element within the operation construct in the portType construct + the input element within the operation construct in the binding construct
- the output element within the operation construct in the portType construct + the output element within the operation construct in the binding construct
- the fault element within the operation construct in the portType construct + the fault element within the operation construct in the binding construct

In other words, if you attach the policy to one of the three message definitions in the abstract or concrete descriptions, you will have attached the expression to the message policy subject.

Again, this makes sense because the two (abstract and concrete) elements represent the same message. But just to hammer this point home, here's one more example:

```
<input>
  <!-- Insert Policy Expression Here -->
  <soap12bind:body use="literal"/>
</input>
```

### WSDL 2.0 Policy Subjects

WSDL 2.0 introduces a few changes as to how attachment points are organized into policy subjects, primarily related to fault definitions (Table 10.1 provides an overview).

WSDL Element	Policy Subject
wsdl20:service	service
wsdl20:endpoint wsdl20:binding wsdl20:interface	endpoint
wsdl20:binding/wsdl20:operation wsdl20:interface/wsdl20:operation	operation
wsdl20:binding/wsdl20:operation/wsdl20:input wsdl20:interface/wsdl20:operation/wsdl20:input	message (for input message)
wsdl20:binding/wsdl20:operation/wsdl20:output wsdl20:interface/wsdl20:operation/wsdl20:output	message (for output message)
wsdl20:binding/wsdl20:fault wsdl20:interface/wsdl20:fault wsdl20:binding/wsdl20:operation/wsdl20:infault wsdl20:binding/wsdl20:interface/wsdl20:infault	message (for an input fault message)
wsdl20:binding/wsdl20:fault wsdl20:interface/wsdl20:fault wsdl20:binding/wsdl20:operation/wsdl20:outfault wsdl20:binding/wsdl20:interface/wsdl20:outfault	message (for an output fault message)

### The `wsp:PolicyReference` Element

The `wsp:PolicyReference` element is the equivalent to an include statement in that it allows you to pull the contents of one policy expression into another. It contains a `URI` attribute that references the `wsu:Id` value of the policy expression to include.

```
<wsp:Policy>
  <wsp:PolicyReference URI="#reliability-policy" />
  <wsam:Addressing>
</wsp:Policy>
</wsam:Addressing>
</wsp:Policy>
```

```
<wsp:Policy wsu:Id="reliability-policy">
  <wsrmp:RMAssertion>
<wsp:Policy/>
</wsrmp:RMAssertion>
</wsp:Policy>
```

In this example, the contents of the policy expression entitled "reliability-policy" are included into the top policy expression, which will result in a composite policy comprised of both `wsrmp:RMAssertion` and `wsam:Addressing` assertion types.

The use of the `wsu:Id` value was introduced at the beginning of this chapter in the *New Namespaces and Prefixes* section. It's a lot like an extensibility attribute in that it allows us to assign an identifier to the policy expression so that it can be referenced elsewhere.

## Embedded Attachments

To embed a policy expression within a WSDL document, we simply add the policy code into the WSDL definitions construct, along with the required namespace, as indicated by the highlighted sections in this example.

```
<definitions targetNamespace=
  "http://actioncon.com/contract/po"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://actioncon.com/contract/po"
  xmlns:po="http://actioncon.com/schema/po"
  xmlns:soap11bind="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:soap12bind="http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns:wsp="http://www.w3.org/2006/07/ws-policy"
  xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
  xmlns:wsrmp="http://docs.oasis-open.org/ws-rx/wsrmp/200702"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
    200401-wss-wssecurity-utility-1.0.xsd">
  ...
  ...
  ...
  <wsp:Policy wsu:Id="composite-policy">
    <wsp:PolicyReference URI="#security-policy"/>
    <wsam:Addressing>
<wsp:Policy/>
</wsam:Addressing>
  <wsrmp:RMAssertion optional="true"/>
  </wsp:Policy>
  <wsp:Policy wsu:Id="security-policy">
    ...
  </wsp:Policy>
</definitions>
```

You might have noticed the following extra line of code in the first embedded policy assertion of the previous example:

```
<wsp:PolicyReference URI="#security-policy"/>
```

Here we can see the `wsp:PolicyReference` element used again as an include mechanism between two separate policy expressions. Although the previous example showed this element incorporate policy content across policy expressions, it is also used when associating inline policy assertions to WSDL elements, as shown in the next example.

```
<binding name="bdPO-SOAP12HTTP" type="tns:ptPurchaseOrder">
  <wsp:PolicyReference URI="#composite-policy"/>
  <soap12bind:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="opSubmitOrder">
    <soap12bind:operation
      soapAction="http://action.com/submitOrder/request"
```

```

        soapActionRequired="true" required="true"/>
    <input>
        <soap12bind:body use="literal"/>
    </input>
    <output>
        <soap12bind:body use="literal"/>
    </output>
</operation>
...
</binding>

```

Here, the WSDL `binding` construct contains an extensibility element comprised of the `wsp:PolicyReference` element that establishes the reference to the policy expression named "composite-policy."

Note that the code for the "composite-policy" expression could have simply been embedded directly within the WSDL `binding` construct. Because the `wsp:PolicyReference` element was used, this expression can now be shared elsewhere within the WSDL definition. For example, you'll notice that this example is comprised of a binding for SOAP 1.2. If the WSDL document also supported SOAP 1.1, the same policy assertion could have been reused by adding the same `wsp:PolicyReference` element to the SOAP 1.1 binding.

### A Complete WSDL Definition with an Attached Policy Expression

One of the reasons that Steve originally designed the Web service contract for the Purchase Order service to include both SOAP 1.1 and 1.2 bindings is that the service could accommodate different types of consumers. Those capable of SOAP 1.2 tend to have more progressive platforms and can therefore support additional WS-\* features.

Most notably, Steve has a requirement for the service to use WS-Addressing headers (explained in Chapter 18). However, after discussions with Kevin, the CTO, there is a reluctance to impose the requirement that all consumers support WS-Addressing. Therefore, Kevin and Steve decide that only those consumers that communicate with the service via SOAP 1.2 must also support the processing of the WS-Addressing headers that they will be designing.

To implement this new requirement, a simple policy expression is created and the WSDL definition is updated as per the highlighted code:

```

<definitions name="Purchase Order"
  targetNamespace="http://actioncon.com/contract/po"
  xmlns:tns="http://actioncon.com/contract/po"
  xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
  xmlns:po="http://actioncon.com/schema/po"
  xmlns:soap11="http://schemas.xmlsoap.org/wSDL/soap/"
  xmlns:soap12="http://schemas.xmlsoap.org/wSDL/soap12/"
  xmlns:wsp="http://www.w3.org/2006/07/ws-policy"
  xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
    200401-wss-wssecurity-utility-1.0.xsd">
  <documentation>
    This is an entity service responsible for purchase
    order-related processing only.
  </documentation>
  <types>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:import namespace=
        "http://actioncon.com/schema/po"
        schemaLocation="http://actioncon.com/schema/po.xsd"/>
    </xsd:schema>
  </types>
  <message name="msgSubmitOrderRequest">
    <part name="PurchaseOrder" element="po:purchaseOrder"/>
  </message>
  <message name="msgSubmitOrderResponse">

```

```

    <part name="Acknowledgement" element="po:acknowledgement" />
</message>
<message name="msgCheckOrderRequest">
  <part name="PONumber" element="po:poNumber" />
</message>
<message name="msgCheckOrderResponse">
  <part name="Status" element="po:status" />
</message>
<message name="msgChangeOrderRequest">
  <part name="PurchaseOrder" element="po:purchaseOrder" />
</message>
<message name="msgChangeOrderResponse">
  <part name="Acknowledgement" element="po:acknowledgement" />
</message>
<message name="msgCancelOrderRequest">
  <part name="PONumber" element="po:poNumber" />
</message>
<message name="msgCancelOrderResponse">
  <part name="Acknowledgement" element="po:acknowledgement" />
</message>
<portType name="ptPurchaseOrder">
  <operation name="opSubmitOrder">
    <input message="tns:msgSubmitOrderRequest" />
    <output message="tns:msgSubmitOrderResponse" />
  </operation>
  <operation name="opCheckOrderStatus">
    <input message="tns:msgCheckOrderRequest" />
    <output message="tns:msgCheckOrderResponse" />
  </operation>
  <operation name="opChangeOrder">
    <input message="tns:msgChangeOrderRequest" />
    <output message="tns:msgChangeOrderResponse" />
  </operation>
  <operation name="opCancelOrder">
    <input message="tns:msgCancelOrderRequest" />
    <output message="tns:msgCancelOrderResponse" />
  </operation>
</portType>
<binding name="bdPO-SOAP11HTTP" type="tns:ptPurchaseOrder">
  <soap11:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="opSubmitOrder">
    <input>
      <soap11:body use="literal" />
    </input>
    <output>
      <soap11:body use="literal" />
    </output>
  </operation>
  <operation name="opCheckOrderStatus">
    <input>
      <soap11:body use="literal" />
    </input>
    <output>
      <soap11:body use="literal" />
    </output>
  </operation>
  <operation name="opChangeOrder">
    <input>
      <soap11:body use="literal" />
    </input>
    <output>
      <soap11:body use="literal" />
    </output>
  </operation>
  <operation name="opCancelOrder">
    <input>
      <soap11:body use="literal" />
    </input>
  </operation>

```

```

    <output>
      <soap11:body use="literal"/>
    </output>
  </operation>
</binding>
<binding name="bdPO-SOAP12HTTP" type="tns:ptPurchaseOrder">
  <wsp:PolicyReference URI="#addressing-policy"/>
  <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/submitOrder/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/submitOrder/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/submitOrder/request"
      soapActionRequired="true" wsdl:required="true"/>
    <input>
      <soap12:body use="literal"/>
    </input>
    <output>
      <soap12:body use="literal"/>
    </output>
  </operation>
</binding>
<service name="svPurchaseOrder">
  <port name="purchaseOrder-http-soap11"
    binding="tns:bdPO-SOAP11HTTP">
    <soap11:address location=
      "http://actioncon.com/services/soap11/po"/>
  </port>
  <port name="purchaseOrder-http-soap12"
    binding="tns:bdPO-SOAP12HTTP">
    <soap12:address location=
      "http://actioncon.com/services/soap12/po"/>
  </port>
</service>
<wsp:Policy wsu:Id="addressing-policy">
  <wsam:Addressing><wsp:Policy/>
</wsam:Addressing>

```

```
</wsp:Policy>  
</definitions>
```

## Conclusion

Polices can turn an ordinary technical interface into a rich, business-centric API capable of enforcing and communicating various types rules and requirements at the contract level. We've only just scratched the surface in this article, as there are many more advanced features provided by the WS-Policy language.

## References

[REF-1] "Web Service Contract Design and Versioning for SOA" by Thomas Erl, Anish Karmarkar, Priscilla Walmsley, Hugo Haas, Umit Yalcinalp, Canyon Kevin Liu, David Orchard, Andre Tost, James Pasley for the "Prentice Hall Service-Oriented Computing Series from Thomas Erl", Copyright Prentice Hall/Pearson PTR and SOA Systems Inc., <http://www.soabooks.com/wsc/>

THE PRENTICE HALL SERVICE-ORIENTED COMPUTING SERIES FROM THOMAS ERL



[Home](#) [Past Issues](#) [Contributors](#) [What is SOA?](#) [SOA Glossary](#) [SOA School](#) [SOA Books](#) [About](#) [Legal](#)

Copyright © 2006-2009  
SOA Systems Inc.