Soya

A Programming Model and Runtime Environment for Component Composition using SSDL

Patric Fornasier, Jim Webber, Ian Gorton

CBSE 2007, Boston, July 2007
Outline

What we will discuss today:

1. SSDL
   (What is it, claimed benefits)
2. Soya – Programming Model
   (Creating SSDL Web Services using C# metadata)
3. Soya – Runtime Environment
   (Protocol validation, correlation, dispatching)
SSDL (SOAP Service Description Language)

What it is...

- **XML** language for describing Web Service contracts
- Assumes **SOAP doc/lit & WS-Addressing**
- Based on **MEST** ideas
- Focusses on **one-way** messages and protocols
- Extensible **protocol** framework
SSDL (SOAP Service Description Language)

Claimed Benefits

- **Simpler** Web Service descriptions
- Fosters creation of **loosely coupled** applications
- Enables **protocol-based** reasoning and integration
SSDL (SOAP Service Description Language)

SSDL Contract Code: Messages

```xml
<ssdl:messages targetNamespace="urn:my:messages" xmlns:s="urn:my:schema">
  <ssdl:message name="MsgA">
    <ssdl:header ref="s:MyHeaderX" mustUnderstand="true" />
    <ssdl:header ref="s:MyHeaderY" role="/ultimateReceiver"/>
    <ssdl:body ref="s:MyBody"/>
  </ssdl:message>
</ssdl:messages>
```
SSDL (SOAP Service Description Language)

SSDL Contract Code: Protocols

```xml
<ssdl:protocol xmlns:mep="urn:ssdl:sc:v1">
  <sc:sc>
    <sc:participant name="ServiceX"/>
    <sc:participant name="ServiceY"/>
    <sc:protocol>
      <sc:sequence>
        <ssdl:msgref ref="MsgA" direction="in" sc:participant="ServiceX"/>
        <sc:choice>
          <ssdl:msgref ref="MsgB" direction="out" sc:participant="ServiceY"/>
          <ssdl:msgref ref="MsgC" direction="out" sc:participant="ServiceY"/>
        </sc:choice>
        <ssdl:msgref ref="MsgD" direction="in" sc:participant="ServiceY"/>
        <ssdl:msgref ref="MsgE" direction="out" sc:participant="ServiceX"/>
      </sc:sequence>
    </sc:protocol>
  </sc:sc>
</ssdl:protocol>
```
More research is needed:

- Lack of **theoretical** and **empirical data**
- Lack of SSDL-aware **middleware**
- Lack of **tool support**
Empirically investigate SSDL in order to:

- **Determine** if the way to describe Web Services implied by SSDL offers significant **benefits**, compared to incumbent approaches
- **Identify** patterns and **best practices** for describing Web Services
- **Provide** design and implementation of a **SSDL runtime**
Programming Model

• Based on Windows Communication Foundation (WCF)
• **Supports** developers to build SSDL services in a straightforward manner
• Encourages creation of service-oriented applications without imposing unrealistic development burdens
• Contractual data defined using **metadata**
Soya Programming Model

Defining Messages

```csharp
[Ssd1MessageContract]
public class MsgA {
    [MessageHeader]
    public string MyHeader;
    [MessageBodyMember]
    public MyData MyBody;
}

[DataContract(Namespace="urn:my:schema")]
public class MyData {
    [DataMember]
    public int id;
    [DataMember]
    public string code;
}
```

```xml
<xs:element name="MyHeader" type="xs:string"/>
<xs:element name="MyBody" type="s:MyData"/>
<xs:complexType name="MyData">
    <xs:sequence>
        <xs:element name="id" type="xs:int"/>
        <xs:element name="code" type="xs:string"/>
    </xs:sequence>
</xs:complexType>

<ssdl:message name="MsgA">
    <ssdl:header ref="s:MyHeader"/>
    <ssdl:body ref="s:MyBody"/>
</ssdl:message>
```
Soya Programming Model

Defining Messaging Behaviour (a.k.a. Protocols)

```csharp
[ServiceContract(Namespace="urn:my:contract")]
[SSdlProtocolContract(Namespace="urn:my:protocol")]
public interface IService {
    [Mep(Style=MepStyle.InOnly)]
    void Process(MsgA msg);

    [Mep(Style=MepStyle.InOut,
        Out=typeof(MsgC), Fault=typeof(FaultX))]
    void Process(MsgB msg);
}
```

```xml
<ssdl:protocol targetNamespace="urn:my:protocol"
        xmlns:mep="urn:ssdl:mep:v1">
    <mep:in-only>
        <ssdl:msgref ref="m:MsgA" direction="in"/>
    </mep:in-only>
    <mep:in-out>
        <ssdl:msgref ref="m:MsgB" direction="in"/>
        <ssdl:msgref ref="m:MsgC" direction="out"/>
        <ssdl:msgref ref="m:FaultX" direction="out"/>
    </mep:in-out>
</ssdl:protocol>
```
Soya

Runtime Environment

• Based on **WCF**
• **Processes** incoming and outgoing SOAP messages
• Ensures **contract conformance** (message structure and ordering)
• Features facilities for message **correlation** and **dispatching**
Building the Runtime

- **Internal service representation**
  - SSDL description
  - Protocol state machine

- **Creation** is protocol-specific and extensible

- **Usage** is protocol-agnostic

Diagram:

1. Reflect over types and attributes
2. Load and apply configuration data
3. Build internal service model and protocol state machine
Runtime Environment

Processing Flow

1. accept messages
2. correlation and state maintenance
3. intercept
4. structural validation
5. protocol validation
6. O/X mapping
7. operation dispatching
8. execute application logic

Service
- Dispatcher
  - OperationInvoker
  - MessageFormatter
  - MessageInspector
  - ContextProvider

WCF Runtime
- Channel Stack
  - protocol
  - encoding
  - transport

Soya Validation R.T.
- XsdValidator
- MessageLocator
- ProtocolStateMachine

WWW
What we have discussed today:

- **SSDL** is a message-centric approach for describing Web Services
  - SOAP & WS-Addressing
  - Protocols
- **Soya** is an SSDL **programming model**
  - C# metadata
- **Soya** is an SSDL **runtime environment**
  - Enacts contract conformance (message structure and ordering)
  - Facilities for correlation and state-based dispatching
  - Generates SSDL that can be exposed to other services
Questions

http://soya.sourceforge.net

http://patforna.blogspot.com