Toward Remote Object Coherence with Compiled Object Serialization for Distributed Computing with XML Web Services

Robert van Engelen and Wei Zhang
Florida State University

Madhusudhan Govindaraju
State University of New York at Binghamton

CPC Jan 10, 2006
Outline

- **Motivation:**
  is object-level coherence in XML Web services feasible?

- **XML serialization:**
  the good, the bad, and the ugly

- **The gSOAP project:**
  a compiled serialization approach for intercommunication

- **Results and conclusions**
Motivation

“Compiler techniques are necessary to improve speed and ensure object coherence in XML-based distributed computing”
Motivation (cont’d)

- Object serialization has increasingly become more powerful and popular as a means to build distributed systems
  - Typically based on remote procedure call (RPC) or its OO equivalent remote method invocation (RMI)
  - XML is gaining popularity as a serialization format

- Message-passing interfaces are also popular, but their low-level APIs are not designed to exchange compound data types and entire object graphs
The Good: a Case for XML Serialization

XML serialization (e.g. used by XML Web services) is gaining popularity, because:

- Binary serialization protocols are inflexible, while XML is an extensible format that supports compositionality
- Binary serialization protocols do not interoperate well across platforms and through firewalls
- Hierarchical business data is easily expressed in XML
- Significant proliferation of XML warehousing
- Newer generation of office tools use XML-based document formats
The Bad: XML Serialization Issues

- XML serialization has shortcomings:
  - How bad does XML serialization impact performance and network bandwidth?
  - “XML = trees”, so how can XML be used to serialize object graphs and ensure object-level coherence?

- Can compiler techniques help to mitigate these concerns?
The Ugly: SOAP RPC Encoding

- Plain-old XML to serialize object graphs won’t work:
  - Naïve approach serializes object graphs into XML trees (this is fine with business data and most numerical data)
  - No standard object referencing mechanism to implement graph edges of object graphs
  - No namespaces to distinguish objects that share the same name but are structurally different

- SOAP RPC encoding standard supports object graphs!
The Ugly (2): SOAP RPC Encoding

- XML namespaces to separate operations and type definitions
- The XSD type system to represent values of primitive types in XML, such as bool, integer, float, string, base64
- A new XML array type to encode sparse and partial arrays
- Id-href XML attributes to implement graph edges to encode multi-ref objects

```xml
<env:Envelope>
  <env:Body env:encodingStyle=""..."">
    <ns:myRemoteOp>
      <arg1 xsi:type="xsd:int">123</arg1>
      <arg2 enc:arrayType="xsd:string[5]">
        <item>abc</item>
        <item href="#_1"/>
        <item href="#_1"/>
        <item enc:position="[5]">xyz</item>
      </arg2>
    </ns:myRemoteOp>
    <mref id="_1">def</mref>
  </env:Body>
</env:Envelope>
```
The Ugly (3): SOAP RPC Encoding

- At first glance this looks promising, but …
  - Serialization is still lossy when floating point values are encoded in text form (due to roundoff)
  - No requirements on multi-ref encoding with id-href: no assurance that the logical structure of an object graph is unchanged at the receiving side
  - Encodes XML with only forward pointing references to multi-referenced nodes (fixed in SOAP 1.2)
The Ugly (4): SOAP Document/literal Style

- More problems with SOAP document/literal style:
  - Document/literal style requires support for the entire XML Schema standard, but XML Schemas are intended for validation and not for data modeling
  - “Impedance mismatch”: programming language type systems are different than XML schema’s type system
  - No default fall-back referencing mechanism such as id-href to implement graph edges, but can use static definition of optional graph edges for each data type
XML Web Services

Client

Object graph x

Send request

Object graph x'

Receive response

Server

Object graph y

Receive request

Object graph y'

Send response

XML Serialization

Client

Server: transform y into y'

XML Deserialization

XML Serialization

XML Deserialization

XML Serialization
Web Service Tools: Lossy Serialization in XML

Serialization in XML can be lossy, because object graph $x$ cannot be recovered from its serialized XML form if we are not careful about the choice of mapping.
Basic Requirement: Object Graph Isomorphism

Object graphs $x$ and $y$ are *isomorph* if $M$ is *bijective* and has an inverse $M^{-1}$

*Object-level coherence* requires bijective mappings, so that distributed and serialized object graphs are always isomorph
Object Coherence Requires Lossless XML Serialization

Serialization is *lossless* if an object graph $x$ can be *recovered* from its serialized XML form $y = M(x)$ by deserializing $x = M^{-1}(y)$

We consider *structural equivalence* only (location in memory is irrelevant) e.g. as in JavaRMI, IIOP, and DCOM
Compiled XML Serialization

Source code type definitions

Compiler

Generates

XML schema definitions

Generates

Points-to analysis

Optimized serializer

XML engine

XML stream

Ptr hash table (graph edges)
Compiled XML Deserialization

XML stream → XML tokenizer → LL(1) parser & deserializer → Pointer remapper

- Source code type definitions
  - generates
  - Compiler
  - Look-aside buffers (cache)
  - Id hash table (resolve id-ref)
# Static Structure Analysis for SOAP RPC Encoding

<table>
<thead>
<tr>
<th>Structure</th>
<th>XML Schema</th>
<th>Example XML Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Diagram" /></td>
<td><img src="image" alt="Diagram" /></td>
<td><img src="image" alt="Diagram" /></td>
</tr>
</tbody>
</table>

**XML Schema**

```xml
<complexType name="X">
  <sequence>
    <element name="y" type="tns:Y"/>
    <element name="z" type="tns:Z"/>
  </sequence>
</complexType>
```

```xml
<x>
  <y href="123"/>
  <y href="123"/>
  <x href="456"/>
  <x href="456"/>
</x>
```
Static Structure Analysis for Document/Literal Encoding

<table>
<thead>
<tr>
<th>Structure</th>
<th>XML Schema</th>
<th>Example XML Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Diagram" /></td>
<td><img src="image" alt="XML Schema" /></td>
<td><img src="image" alt="XML Instance" /></td>
</tr>
</tbody>
</table>

```
<complexType name="X">
  <sequence>
    <element name="y" type="tns:Y"/>
    <element name="z" type="tns:Z"/>
  </sequence>
</complexType>
```

```
<x>
  <y>...</y>
  <z>...</z>
  ...
</x>
```

```
<x ref="123">...</x>
  ...
  <x ref="123">...</x>
  ...
  <y id="123">...</y>
```

```
<x ref="456">...</x>
  ...
  <x ref="456">...</x>
  ...
  <x id="456">...</x>
```
Static Structure Analysis for Compiled XML Serialization

- Construct a data model
  - Compiler determines which data type instances can refer to other data type instances at run time

- Generate type-specific points-to analysis algorithm
  - Only needed for pointer types and structs/classes with pointer fields

- Generate type-specific optimized serialization code
  - Serializer uses id-ref linking based on pointer analysis
Constructing a Plausible Data Model from Source Code

typedef int SSN;

struct Node
{
    int val;
    int *ptr;
    float num;
    struct Node *next;
};

Source code type definitions

Data model graph
(arcs denote all possible pointer references)
Generating an XML Schema Definition for Serialized XML

```
<complexType name="Node">
  <sequence>
    <element name="val" type="int"/>
    <element name="ptr" type="int" minOccurs="0"/>
    <element name="num" type="float"/>
    <element name="next" type="tns:Node" minOccurs="0"/>
  </sequence>
</complexType>

<simpleType name="SSN">
  <restriction base="int"/>
</simpleType>
```

Data model graph
Generating Code for Runtime Points-To Analysis

```c
serialize_pointerToInt(int *p)
{
    if (p != NULL)
    {
        // lookup and mark (p,TYPE_int)
        // as target in ptr hash table
    }
}

serialize_Node(struct Node *p)
{
    if (p != NULL)
    {
        // lookup and mark (p,TYPE_Node)
        // as target in ptr hash table
        mark_embedded(&p->val,TYPE_int);
        serialize_pointerToInt(p->ptr);
        // skip p->num
        serialize_pointerToNode(p->next);
    }
}
```

Data model graph
Generating Serialization Code

```c
put_pointerToInt(int *p)
{
    if (p != NULL)
    {
        // lookup (p, TYPE_int)
        // if embedded, then output "ref"
        // if single, then output value
        // if multi, then output value
        // with "id" and mark embedded
    }
}

put_Node(struct Node *p)
{
    if (p != NULL)
    {
        // lookup (p, TYPE_Node)
        // if embedded, then output "ref"
        // if single, then output value
        // if multi, then output value
        // with "id" and mark embedded
        put_int(&p->val);
        put_pointerToInt(p->ptr);
        put_float(&p->num);
        put_pointerToNode(p->next);
    }
}
```
Serialization Example

Ptr hash table after runtime points-to analysis by the generated algorithm constructed from the data model

<table>
<thead>
<tr>
<th>ID</th>
<th>PtrLoc</th>
<th>PtrType</th>
<th>PtrSize</th>
<th>PtrCount</th>
<th>RefType</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>Node</td>
<td>16</td>
<td>2</td>
<td>multi</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>int</td>
<td>4</td>
<td>1</td>
<td>embedded</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>Node</td>
<td>16</td>
<td>1</td>
<td>single</td>
</tr>
<tr>
<td>4</td>
<td>C</td>
<td>int</td>
<td>4</td>
<td>1</td>
<td>single</td>
</tr>
</tbody>
</table>

CPC Jan 10, 2006 24
Example (cont’d)

Node loc=A
val=123 ptr=B num=1.4 next=B

embedded, id=2

single

Node loc=B
val=456 ptr=C num=2.3 next=A

multi, id=1

Node loc=C
val=789

multi, id=1

single
Example (cont’d)

<Node id="_1">  
  <Node id="1">  
  </Node>  
</Node>
Example (cont’d)

```xml
<Node id="_1">
    <val>123</val>
</Node>

<SSN loc=C>
    =789
</SSN>
```
Example (cont’d)

```
<Node id="_1">
  <val>123</val>
  <ptr ref="#_2"/>
</Node>
```

```xml
<Node id="_2">
  <val>123</val>
  <ptr ref="#_2"/>
</Node>
```
Example (cont’d)

```
<Node id="_1">
  <val>123</val>
  <ptr ref="#_2"/>
  <num>1.4</num>
</Node>
```

```
<Node id="_2">
  <val>456</val>
  <ptr ref="#_3"/>
  <num>2.3</num>
</Node>
```

```
<Node id="_3">
  <val>789</val>
  <ptr ref="#_4"/>
  <num>1.4</num>
</Node>
```

SSN
loc=C
Example (cont’d)

<Node id="_1">
  <val>123</val>
  <ptr ref="#_2"/>
  <num>1.4</num>
  <next>
    <Node loc=B>
      <val>456</val>
      <ptr>“C”
      <num>2.3</num>
      <next>
        <Node loc=A>
          <val>789</val>
          <SSN>loc=C</SSN>
        </Node>
      </next>
    </Node>
  </next>
</Node>
Example (cont’d)

```xml
<Node id="_1">
  <val>123</val>
  <ptr ref="#_2"/>
  <num>1.4</num>
  <next>
    <val id="_2">456</val>
  </next>
</Node>

<Node id="_2">
  <val>456</val>
  <ptr ref="#_3"/>
  <num>2.3</num>
  <next>
    <val>789</val>
  </next>
</Node>
```

multi, id=1

embedded, id=2

single

multi, id=1

single

SSN
loc=C

Node
loc=B

Node
loc=A
Example (cont’d)

```xml
<Node id="_1">
  <val>123</val>
  <ptr ref="#_2"/>
  <num>1.4</num>
  <next>
    <val id="_2">456</val>
    <ptr>789</ptr>
  </next>
</Node>
```

CPC Jan 10, 2006
Example (cont’d)

```
<Node id="_1">
  <val>123</val>
  <ptr ref="#_2"/>
  <num>1.4</num>
  <next>
    <val id="_2">456</val>
    <ptr>789</ptr>
    <num>2.3</num>
  </next>
</Node>
```

CPC Jan 10, 2006
Example (cont’d)

```
<Node id="_1">
    <val id="_2">123</val>
    <ptr>789</ptr>
    <num>1.4</num>
    <next ref="#_1" />
</Node>
```

```xml
<Node id="_1">
    <val>123</val>
    <ptr ref="#_2" />
    <num>1.4</num>
    <next>
        <val id="_2">456</val>
        <ptr>789</ptr>
        <num>2.3</num>
        <next ref="#_1" />
    </next>
</Node>
```

CPC Jan 10, 2006
Example (cont’d)

```xml
<Node id="_1">
  <val>123</val>
  <ptr ref="#_2"/>
  <num>1.4</num>
  <next>
    <val id="_2">456</val>
    <ptr>789</ptr>
    <num>2.3</num>
    <next ref="#_1"/>
  </next>
</Node>
```

Node loc=A, multi, id=1

Node loc=B, single

Node loc=C, SSN

val=123, ptr=B, num=1.4, next=B

val=456, ptr=C, num=2.3, next=A

val=456, ptr=C, num=2.3, next=A

val=789, ptr=B, num=1.4, next=B

val=456, ptr=C, num=2.3, next=A
Compiled XML Deserialization

- For each data type, generate optimized deserialization code:
  - Generate a recursive descent LL(1) XML parser that is optimized for the data type
  - Embed deserialization operations as semantic actions in the parser
  - Invoke id hash table lookups as semantic actions to resolve id-ref references on the fly
- The pointer remapper ensures the consistency of pointers in the object graph when nodes are moved in the construction process
Deserialization Example

```xml
<Node id="_1">
  <val>=?  <ptr>=?  <num>=?  <next>=?
</Node>
```

Recursive descent parsing with object deserialization operations as semantic actions.
Example (cont’d)

<Node id="1">
  <val>123</val>
</Node>

Recursive descent parsing with object deserialization operations as semantic actions
Example (cont’d)

<Node id="_1">
  <val>123</val>
  <ptr ref="#_2"/>
</Node>

Recursive descent parsing with object deserialization operations as semantic actions
Example (cont’d)

```xml
<Node id="_1">
  <val>123</val>
  <ptr ref="#_2"/>
  <num>1.4</num>
</Node>
```

Recursive descent parsing with object deserialization operations as semantic actions
Example (cont’d)

<Node id="_1">
    <val>123</val>
    <ptr ref="#_2"/>
    <num>1.4</num>
    <next/>
</next>
</Node>

Recursive descent parsing with object deserialization operations as semantic actions
Example (cont’d)

```xml
<N ode id="_1">
  <val>123</val>
  <ptr ref="#_2"/>
  <num>1.4</num>
  <next>
    <val id="_2">456</val>
  </next>
</Node>
```

Recursive descent parsing with object deserialization operations as semantic actions
Example (cont’d)

<Node id="_1">
  <val>123</val>
  <ptr ref="#_2"/>
  <num>1.4</num>
  <next>
    <val id="_2">456</val>
    <ptr>789</ptr>
  </next>
</Node>

Recursive descent parsing with object deserialization operations as semantic actions

![Diagram showing nodes and values]
<Node id="_1">
  <val>123</val>
  <ptr ref="#_2"/>
  <num>1.4</num>
  <next>
    <val id="_2">456</val>
    <ptr>789</ptr>
    <num>2.3</num>
  </next>
</Node>

Recursive descent parsing with object deserialization operations as semantic actions
Example (cont’d)

Recursive descent parsing with object deserialization operations as semantic actions
Example (cont’d)

```xml
<Node id="_1">
  <val>123</val>
  <ptr ref="#_2"/>
  <num>1.4</num>
  <next>
    <val id="_2">456</val>
    <ptr>789</ptr>
    <num>2.3</num>
    <next ref="#_1"/>
  </next>
</Node>
```

Recursive descent parsing with object deserialization operations as semantic actions
Implementation: the gSOAP Toolkit

- Web service applications are built in two stages:
  1. Generate service definitions in familiar C/C++ header file format
  2. Generate client/server stubs and skeleton source code and serialization code
- Note: the soapcpp2 compiler can also be used to generate WSDL from header files
1. Analyze WSDL/Schema and Generate C/C++ Header File

Service defs: service.wsdl

wsdl2h tool

Header file defs: service.h

Documented methods and class diagrams
2. Analyze Header File and Generate Stubs and Serializers

Header file defs:
- service.h

soapcpp2 compiler

soapClient.cpp
soapServer.cpp
soapC.cpp

gSOAP application

client/server API

LL(1) XML parsers and (de)serializers

schema-opt serializer
schema-opt deserializer

gSOAP engine
HTTP/S + TCP/IP + UDP

XML data

Network fabric

Native C/C++ data

Plugins
- stats
- logging
- WSSE
### Performance Results

- **gSOAP serialization+deserialization perf. with 1.2K structures**

<table>
<thead>
<tr>
<th>calls per second</th>
<th>Operating system</th>
<th>gSOAP, compiler</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>3241</td>
<td>Linux 2.6.5 64-bit</td>
<td>gSOAP 2.5, gcc 3.3.3 -02 64-bit</td>
<td>1x AMD FX-53 2.4GHz</td>
</tr>
<tr>
<td>2940</td>
<td>Linux 2.6.5 64-bit</td>
<td>gSOAP 2.7.0c, gcc 3.3.3 -02 64-bit</td>
<td>1x AMD Opteron 148 2.2GHz</td>
</tr>
<tr>
<td>2820</td>
<td>Linux 2.6.5 64-bit</td>
<td>gSOAP 2.5, gcc 3.3.3 -02 64-bit</td>
<td>1x AMD Opteron 148 2.2GHz</td>
</tr>
<tr>
<td>264</td>
<td>Linux 2.6.5 64-bit</td>
<td>gSOAP 2.5, gcc 3.3.3 -02 64-bit</td>
<td>1x AMD Opteron 148 2.2GHz</td>
</tr>
<tr>
<td>248</td>
<td>Linux 2.6.5 64-bit</td>
<td>gSOAP 2.5, gcc 3.3.3 -02 64-bit</td>
<td>1x AMD Opteron 148 2.2GHz</td>
</tr>
<tr>
<td>2340</td>
<td>Linux 2.6.5 64-bit</td>
<td>gSOAP 2.5, gcc 3.3.3 -02 64-bit</td>
<td>1x AMD Opteron 148 2.2GHz</td>
</tr>
<tr>
<td>2130</td>
<td>Linux 2.6.5 64-bit</td>
<td>gSOAP 2.5, gcc 3.2.2 -02 64-bit</td>
<td>2x AMD Opteron 244 1.8GHz</td>
</tr>
<tr>
<td>2280</td>
<td>Linux 2.6.5 64-bit</td>
<td>gSOAP 2.5, gcc 3.2.2 -02 64-bit</td>
<td>2x AMD Opteron 244 1.8GHz</td>
</tr>
<tr>
<td>2070</td>
<td>Linux 2.6.9 IA-64</td>
<td>gSOAP 2.5, Intel ia64 8.1 -02</td>
<td>2x Harnium2 1.4GHz</td>
</tr>
<tr>
<td>1930</td>
<td>Linux 2.6.5</td>
<td>gSOAP 2.5, gcc 3.4.0 -03</td>
<td>1x Pentium 3 3.0GHz (w/o HT)</td>
</tr>
<tr>
<td>1760</td>
<td>Linux 2.4.23</td>
<td>gSOAP 2.5, gcc 3.3.3 -02</td>
<td>1x Pentium4 3.0GHz (w/o HT)</td>
</tr>
<tr>
<td>1550</td>
<td>Linux 2.4.23</td>
<td>gSOAP 2.5, gcc 3.2.2 -02</td>
<td>2x Xeon P4 3.0GHz (w/o HT)</td>
</tr>
<tr>
<td>1310</td>
<td>Linux 2.4.23</td>
<td>gSOAP 2.5, gcc 3.1.1 -02</td>
<td>2x Xeon P4 3.0GHz (w/o HT)</td>
</tr>
<tr>
<td>1180</td>
<td>Linux 2.4.19 IA-64</td>
<td>gSOAP 2.5, Intel ia64 7.0 -02</td>
<td>2x Harnium 1GHz</td>
</tr>
<tr>
<td>1040</td>
<td>Linux 2.4.19 IA-64</td>
<td>gSOAP 2.5, gcc 2.96 -02</td>
<td>2x Harnium 1GHz</td>
</tr>
<tr>
<td>973</td>
<td>Linux 2.4.23</td>
<td>gSOAP 2.5, gcc 2.954</td>
<td>1x Pentium4 2.5GHz</td>
</tr>
<tr>
<td>830</td>
<td>Linux 2.4.23</td>
<td>gSOAP 2.5, gcc 2.954</td>
<td>1x Pentium4 2.5GHz</td>
</tr>
<tr>
<td>703</td>
<td>Linux 2.4.23</td>
<td>gSOAP 2.5, gcc 2.90615 -02</td>
<td>2x Power4+ 1.2GHz</td>
</tr>
<tr>
<td>515</td>
<td>Linux 2.4.23</td>
<td>gSOAP 2.5, gcc 2.8.1 -02</td>
<td>2x UltraSPARC-III 750MHz</td>
</tr>
</tbody>
</table>

**3241 calls per second**
Performance Comparison to Other XML Parsers

![Graph comparing XML parser performance](image)

- **Experimental parsers**
- **CPC Jan 10, 2006**
Conclusions

Is XML serialization a viable alternative to binary serialization protocols?

+ It has the advantages of cross-platform interoperability and flexibility to manipulate XML with tools such as XSLT, filters, data mining, persistent storage, etc.

− Disadvantage is that naïve serialization can be lossy, implying object-level coherence issues

− Disadvantage is the low performance of XML parsers

+ However, compiled serialization speedup is significant