Data Representation and Remote Procedure Calls

Srinidhi Varadarajan
Topics

- External data representation
  - Motivation
  - Approaches
  - NDR, ASN.1, and XDR

- Remote procedure calls
  - Concepts
  - ONC RPC
    - General operation
    - Code example
Need for Data Representation (1)

- Network applications pass many types of data
  - Characters and character strings
  - Integers (of different lengths)
  - Floats (of different lengths)
  - Arrays and structures (flat types)
  - Complex types (using pointers)

- Different host architectures may use different internal representations
  - Networked environments are often heterogeneous
Need for Data Representation (2)

- Example: \((300)_{10} = (13C)_{16}\)
  - Stored as a long integer: 00 00 01 3C
  - “Big endian” versus “little endian”

<table>
<thead>
<tr>
<th></th>
<th>Big endian</th>
<th>Little endian</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte i:</td>
<td>00</td>
<td>3C</td>
</tr>
<tr>
<td>byte i+1:</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>byte i+2:</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>byte i+3:</td>
<td>3C</td>
<td>00</td>
</tr>
</tbody>
</table>
Potential Solutions (1)

- Asymmetric conversion
  - Convert at one end (client or server)
  - Must know the host type of destination or source
  - With N types of hosts, need N(N-1) converters total.
  - Sometimes known as “receiver-makes-right”
  - Basis for NDR
Potential Solutions (2)

- Symmetric conversion
  - Convert to and from a *canonical intermediate form* -- an external data representation
  - Flexible and portable, but at a cost in computation
    - Conversion required even if client and server use the same internal representation
  - With N types of hosts, requires 2N converters
    - Fewer converters than for asymmetric conversion
    - But, N is usually small
  - Basis for XDR and ASN.1
Potential Solutions (3)

- Symmetric conversion (continued)

*big endian*

client data

convert

htonl()

*little endian*

server data

convert

ntohl()
Network Data Representation (1)

- NDR is used in the Distributed Computing Environment (DCE)
- Uses asymmetric “receiver-makes-right” approach
- Format
  - Architecture tag at the front of each message
    - “Big endian” or “little endian”
    - ASCII or EBCDIC
    - IEEE 754 or other floating point representation
Network Data Representation (2)

- **Architecture tag**

<table>
<thead>
<tr>
<th></th>
<th>4</th>
<th>4</th>
<th>8</th>
<th>8</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integr Rep</td>
<td>Char Rep</td>
<td>Float Rep</td>
<td>Extension 1</td>
<td>Extension 2</td>
<td></td>
</tr>
</tbody>
</table>

4/4/2001
Abstract Syntax Notation One (1)

- **ASN.1** is an ISO standard
  - Scope is broader than network data representation
  - Basic Encoding Rules (BER) defines representation
- **Uses a canonical intermediate form** *(symmetrical)*
- **Uses a triple to represent each data item**
  - `< tag, length, value >`
  - Tag defines type (usually 8 bits)
  - Length is number of bytes in value field
  - Value is in canonical intermediate form
Abstract Syntax Notation One (2)

- Example

```
<table>
<thead>
<tr>
<th>typ</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>4</td>
</tr>
</tbody>
</table>
```

- Compound data types can be represented by nesting primitive types

value
Abstract Syntax Notation One (3)

- Length field can be made arbitrarily large
  - 1- to 127-byte value

Greater than a 127-byte value

```
length
```

```
1       n
```

\( n \) bytes containing length
External Data Representation (1)

- XDR is used with SunRPC (Open Network Computing RPC)
  - Defined in RFC 1014
- Uses a canonical intermediate form (symmetrical)
- Types are implicit
  - XDR codes data, but not the type of data
  - Type of data must be determined by application protocol
- Tags are not used except to indicate array lengths
Example XDR encoding of a structure

```c
struct example {
    int count;
    int values[2];
    char buffer[4];
}
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>2 450</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>898</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D</td>
</tr>
</tbody>
</table>
Creating an XDR Data Stream (1)

1) Create buffer
   - `xdrmem_create(xdrs, buf, BUFSIZE, XDR_ENCODE);

2) Make calls to build buffer
   - `int i = 300;
     xdr_int(xdrs, &i);`
Creating an XDR Data Stream  (2)

- Sample routines (see fig 20.4 in text)
  - xdr_bool()
  - xcr_bytes()
  - xdr_enum()
  - xdr_float()
  - xdr_vector()
  - xdr_string()
  - xdr_opaque()

- Same calls are used to encode and decode

- Stream header specifies direction
  - For decode: xdrmem_create(xdrs, buf, BUFSIZE, XDR_DECODE);
Comparing XDR, ASN.1, and NDR

- Symmetric versus asymmetric trade-off for comparing ASN.1 and XDR to NDR
  - Potentially more converters needed for NDR, but number of different host types is small
  - Overhead of type fields
  - Conversion can often be avoided

- Comparing ASN.1 and XDR
  - XDR has less overhead than ASN.1 since it does not use tags
  - XDR adheres to natural byte boundaries
  - Expressiveness of ASN.1 is very rich, more flexible than XDR
Remote Procedure Calls

- Remote Procedure Call (RPC) is an alternate model for networked applications
- Used for many standard applications
  - NFS
  - NIS, NIS+
  - Microsoft Exchange Server
  - and others ...
- Closely associated with data representation
  - Function parameters must pass over the network
Models for Distributed Applications

- Communication-oriented design
  - Focus on protocol and communications
  - Our approach to date
- Application-oriented design
  - Focus on application program structure and make communications “transparent”
  - RPC approach
A Traditional Program (1)
A Traditional Program (2)
Make the Program Distributed (1)

- proc1, proc4, and proc5 are remote procedures
Make the Program Distributed (2)

- **Call** -- send message to invoke remote procedure
- **Return** -- send reply back to client

```
main
  proc 1
  call proc1
  call proc4
  return
  return
exit
```
RPC Components

- **Call P**
- **Client Stub**
- **RPC**
- **Marshaled Arguments**
- **Server Stub**
- **RPC**
- **Message**
- **Interface Description for P**
- **Stub Compiler**
- **Proc P**

- Code code code
- Code code code
Marshaling Arguments

application
data structure

marshaling
RPC Design Issues

- Control is multithreaded
  - Procedures executed on different hosts
  - Different threads for each call
- No shared memory
- No shared resources, e.g. files
- More arguments
  - Since no shared memory or other resources
- Server must be active or can be invoked
- Message interface
Open Network Computing (ONC) RPC

- Developed by Sun Microsystems

“Remote programs”

- Remote procedures plus shared global data
- Not just remote procedure

Functionality

- Message formats -- carried by TCP or UDP
  - Pass arguments, results, other information
- Naming scheme for remote programs and procedures
  - Program, version, procedure
- Authentication scheme
ONC RPC Communications

- Can use TCP or UDP
  - RPC does nothing itself to provide reliability

- With UDP ...
  - If client receives a reply, then “at least once” semantics apply
  - If client does not receive a reply, then “zero or more” semantics apply
  - Must be considered in design
    - “read 20 bytes starting at 100”, not
    - “read the next 20 bytes”

- With TCP ...
  - Reliable due to use of TCP
Port Mapper (1)

- “Port mapper” allows dynamic mapping between protocol port numbers and remote programs
- Remote programs (servers) register with the port mapper on their local host
- Clients query port mapper at well-known port number (111) to get port for remote program
Port Mapper (2)

1. register(prog, vers, port)
2. request(prog, vers)
3. reply(port)
4. call/return

RPC Program → Port Mapper
RPC Client

remote program port → well known port → client port

RPC
Program
Port
Mapper
RPC
Client

4/4/2001
Stub Routines (1)

- Traditional program to be partitioned

```
proc A

proc B

proc C
```

Local

Remote
Stub Routines (2)

- After partitioning with stub routines

proc A

proc B
client stub

proc B
server stub

proc C
client stub

proc C
server stub

Local (Client)

Remote (Server)
Client Stub

- Is called by client program
- “Marshals” arguments
  - XDR used to encode (with ONC RPC)
- Sends CALL to server
- Waits for reply
- “De-marshals” arguments
  - XDR used to decode
- Returns to client program
  - Client just makes a call that then returns
Server Stub

- Is dispatched
- Accepts arguments, de-marshals and decodes with XDR
- Calls server program procedure
- Procedure returns to stub
  - Server procedure is just called and later returns
- Marshals results and encodes with XDR
- Sends results back to client
- Exits
Dispatcher

proc A

proc B client stub

proc B server stub

dispatcher

call

proc C client stub

proc C server stub

proc B

proc C
RPCGEN

- RPCGEN is the RPC program “generator”
- Simplifies the creation of a distributed application using RPC
- Input descriptions of ...
  - Remote procedures and interfaces
  - User-defined data types, e.g. structures
- Output files ...
  - Client and server stub files
  - Conversion routines for user-defined data types
  - Common header file
Code Generation using RPCGEN

```
4/4/200137
oncrpc.lib
compile
rdict_srp.c
rdict_sif.c

client
compile
oncrpc.lib
server
rdict_xdr.c
rdict.h
rdict_svc.c

rdict.x
rpcgen

dict_svc.c
```
ONC RPC Code Example Files

- rdict.x: interfaces, common values, data structures
- rdict.h: common header file
- rdict_xdr.c: XDR translations
- rdict_clnt.c: sends calls from client to server
- rdict_svc.c: dispatcher, sends calls from server to client
- rdict.c: main client
- rdict_cli.c: client stub procedures
- rdict_srp: main server routines
- rdict_sif.c: server stub procedures
ONC RPC Code Example Call Sequence

\[
\text{\texttt{insert}} \rightarrow \text{\texttt{main()}} \quad \text{rdict.c} \\
\text{\texttt{insertw()}} \quad \text{rdict_cif.c} \\
\text{\texttt{insertw_1()}} \quad \text{rdict_clnt.c} \\
\text{\texttt{clnt_call()}} \quad \text{oncrpc.lib} \\
\text{\texttt{svc_run()}} \quad \text{oncrpc.lib} \\
\text{\texttt{svc_sendreply()}} \quad \text{oncrpc.lib} \\
\text{\texttt{rdictprog_1()}} \quad \text{rdict_svc.c} \\
\text{\texttt{insertw_1()}} \quad \text{rdict_sif.c} \\
\text{\texttt{insertw_1()}} \quad \text{rdict_srp.c} \\
\text{\texttt{insertw()}} \\
\]
You should now be able to … (1)

- Describe different schemes for data representation and identify strengths and weaknesses
  - Generic models
  - Specific schemes (NDR, ASN.1, XDR)
- Show how simple data types would be represented using NDR, ASN.1, and XDR
- Describe the structure of an RPC application including role of stub procedures
- Describe the need for marshaling and where marshaling is implemented
You should now be able to … (2)

- Describe the structure and operation of …
  - ONC RPC
- Define the role of …
  - RPCGEN
- Design and analyze simple applications using ONC RPC