Issues in Design and Implementation of Serialization

Instructor: Prasun Dewan (FB 150, dewan@unc.edu)
```java
ObjectOutputStream socketOut = new ObjectOutputStream(socket.getOutputStream());
socketOut.writeObject(list);
ObjectInputStream socketIn = ObjectInputStream(socket.getInputStream());
List readVal = (List) socketIn.readObject();
```
BASIC Serialization

- Handle Header (Node, ByteSequence)
  - ∀component, C
    - Handle Component Header (C, ByteSequence)
  - Visit(C, ByteSequence)
    - Atomic (C)?
      - Handle Atomic (C, ByteSequence)
  - Visit(Node, ByteSequence)
**Serialization vs. Deserialization**

- **Handle Header (Node, ByteSequence)**
- **Visit(Node, ByteSequence)**
- **∀ component, C**
- **Handle Component Header (C, ByteSequence)**
- **Visit(C, ByteSequence)**
- **Atomic (C)?**
- **Handle Atomic (C, ByteSequence)**

*Assuming that ByteSequence is a Reference Parameter*

*Visit would write or read the value depending on the mode*
Basic Serialization

Handle Header (Node, ByteSequence)

∀component, C

Handle Component Header (C, ByteSequence)

Visit(Node, ByteSequence)

Visit(C, ByteSequence)

Atomic (C)?)

Handle Atomic (C, ByteSequence)

Header?
IMPLEMENTING Serialization

The other party must be able to create an instance of the appropriate type based on the originator’s type.
**Nature of Serialized Message**

- **Type (ID)**
  - Serialization

- **Value**
  - Serialization

**Type ID** can be string or some other well-defined identifier

**Type is optional**

**Class**

```java
newInstance()
```
**When is Type ID Needed**

<table>
<thead>
<tr>
<th>Type needed when other party does not know Atomic type or class of received value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port or application may constrain values to expected types</td>
</tr>
<tr>
<td>In remote assignment or procedure call, types well defined</td>
</tr>
<tr>
<td><code>send load(2.2)</code></td>
</tr>
<tr>
<td><code>send add (5, 3)</code></td>
</tr>
</tbody>
</table>

Nature of value representation?
ATOMIC?

Handle Header (Node, ByteSequence)

∀component, C

Handle Component Header (C, ByteSequence)

Visit(Node, ByteSequence)

Visit(C, ByteSequence)

Atomic (C)?

Handle Atomic (C, ByteSequence)

Handle Atomic?
**Atomic Values**

An atomic has a direct XDR that does not have to be composed from other XDR's.

```
int s = 3
```

**Little Endian**

- x86
  - 0
  - 3

**Big Endian**

- IBM 370
  - 0
  - 3

Host-independent eXternal Data Representation

Atomic, Wrapper
**Atomic Binary**

- **Little Endian**
  - x86
  - 0
  - 3

- **Big Endian**
  - IBM 370
  - 3
  - 0

**Network/External value representation:** Some agreed upon binary representation of value in a sequence of bytes
Little
Endian
x86

{0, 3}

int s = 3

“3”

{3, 0}

Big
Endian
IBM 370

Textual: A standard string representation of value
# Textual vs. Binary Atomic Values

Inefficient

<table>
<thead>
<tr>
<th>Human readable: debugging and logging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consistent with Composite Object Serialization for Heterogeneous Platforms</td>
</tr>
<tr>
<td>Can be sent over SMTP and HTTP (XML Serialization, SOAP, Web Services)</td>
</tr>
</tbody>
</table>
SMTP for IPC?

- Slow – many Handlees involved on path
- No connection set up
- Sometimes speed not necessary
- Chess program
- Human to Handle communication
- Invoke remote service with appropriately formatted email
Atomic Serialization

Type (Id Serialization)  Atomic XDR
COMPONENTS?

- Handle Header (Node, ByteSequence)
  - ∀component, C
    - Handle Component Header (C, ByteSequence)

- Visit(Node, ByteSequence)
  - Visit(C, ByteSequence)
    - Atomic (C)?
      - Handle Atomic (C, ByteSequence)
  - ∀component, C?
WHAT ARE COMPONENTS

∀component, C

Physical Structure: Memory representation

Logical Structure: Interface + Pattern
PROGRAMMER-DEFINED CLASS:
ABMISpreadsheet

```java
public class ABMISpreadsheet implements BMISpreadsheet {
    double height = 1.77;
    double weight = 75;
    public double getWeight() {return weight;}
    public void setWeight(double newWeight) {
        weight = newWeight;
    }
    public double getHeight() {return height;}
    public void setHeight(double newHeight) {height = newHeight;}
    public double getBMI() {return weight/(height*height);}
}
```
**Physical Structure**

```
BMISpreadsheet b = new ABMISpreadsheet()
```

- **ABMISpreadsheet**
  - **height**
    - double 1.77
  - **weight**
    - double 75.0

- Physical structure defined by instance variables stored in memory
- Internal nodes labeled by class name
- Each component of an object is a value stored in an instance variable declared in class
- Leaf nodes labeled type and value
Logical Structure

```java
BMISpreadsheet b = new AnUnencapsulatedBMISpreadsheet()
```

```java
AnUnencapsulatedBMISpreadsheet

height

bmi

weight

double 1.77
double 23.9
double 75.0
```
**Bean Pattern**

Typed, Named Unit of Exported Object State

```java
public class C {
    public T getP() {
        ...
    }
    public void setP(T newValue) {
        ...
    }
}
```

- **Name P**
- **Type T**
- **Read-only**
- **Editable**
- **Getter method**
- **Setter method**

**Bean convention:**
For humans and tools

Violation: Obtaining `P` directly violates Bean convention.
Physical Structure

```java
IntHistory ints = new AnIntHistory();
ints.add(3)
```
IntHistory ints = new AnIntHistory();
ints.add(3)

Can define Bean-like patterns for Lists (and Tables)

Vector and ArrayList follow these patterns but do not notify
public class AnotherIntHistory implements IntHistory {
    List<Integer> contents = new ArrayList();
    public int size() {
        return contents.size();
    }
    public int get (int index) {
        return contents.get(index);
    }
    public void add (int element) {
        contents.add(element);
    }
}
PHYSICAL STRUCTURE

```java
IntHistory ints = new AnotherIntHistory();
ints.add(3)
```
IntHistory ints = new AnotherIntHistory();
ints.add(3)
CONVENTIONS FOR VARIABLE-SIZED COLLECTION

public interface C{
    public T elementAt(int index);
    public int size();
    public Any setElementAt(T t, int index);
    ...
}

Arbitrary Type.

Write method

Convention based on Vector

Unconstrained Type (void or T in practice)

Read methods
How to Derive Components?

∀ component, C

Physical Structure: Memory representation

Logical Structure: Interface + Pattern
HOW TO DETERMINE PHYSICAL COMPONENTS

Array elements can be directly accessed

Need to determine instance variables of programmer defined objects

Can we do this in a replaceable library?
Field Reflection

Class
- Field[] getFields()
- Field[] getDeclaredFields()

Field
- String getName()
- Object get(Object parent)
- Object set(Object parent, Component c)
- void setAccessible(boolean b)

All public
- All non-inherited

Serialization
- DeSerialization

Can break encapsulation!

Logical components?
**Method Reflection**

**Class**
- getMethods()

**Method**
- execute (targetObject, params)
- getParameterTypes()
- getReturnType()
- getName()

(De)Serialization

Provides reflection operations to learn properties of the action such as return type, name, and parameter types.
public interface BMISpreadsheet {
    double getWeight();
    void setWeight(double newWeight);
    double getHeight();
    void setHeight(double newHeight);
    double getBMI();
}
public static void main(String[] args) {
    BMISpreadsheet bmi = new ABMISpreadsheet();
    printProperties(bmi);
}

printProperties() accepts an argument of arbitrary type
public static void printProperties(Object object) {
    System.out.println("Properties of:" + object);
    Class objectClass = object.getClass();
    Method[] methods = objectClass.getMethods();
    Object[] nullArgs = {};
    for (int index = 0; index < methods.length; index++) {
        Method method = methods[index];
        if (isGetter(method)) {
            Object retVal = methodInvoke(object, method, nullArgs);
            System.out.println(propertyName(method) + ":" + retVal);
        }
    }
    System.out.println();
}

Class is a runtime variable vs. compile time as in generics.
Invoking methods on a class to create learn its properties.
public static String GETTER_PREFIX = "get";

public static boolean isGetter (Method method) {
    return method.getParameterTypes().length == 0 &&
        method.getReturnType() != Void.TYPE &&
        method.getName().startsWith(GETTER_PREFIX);
}

public static String propertyName (Method getter) {
    return getter.getName().substring(GETTER_PREFIX.length());
}

**Invoking Target Method**

```java
public static Object methodInvoke(Object object, Method method, Object[] args) {
    try {
        return method.invoke(object, args);
    } catch (IllegalAccessException e) {
        e.printStackTrace();
        return null;
    } catch (InvocationTargetException e) {
        e.printStackTrace();
        return null;
    }
}
```

**IllegalAccessException**: Method not visible to caller.

**InvocationTargetException**: Exception thrown when parameters/target object do not match method or method throws exception such as ClassCastException.

Method that takes method as a parameter is 2nd-order method.
Method Reflection

Class
- getMethods()

Method
- execute (targetObject, params)
- getParameterTypes()
- getReturnType()
- getName()

Provides reflection operations to learn properties of the action such as return type, name, and parameter types.

Higher level abstraction?
INTROSPECTION

Identifies directly the components defined by some pattern

Provides a way to invoke read and write operations on the components

Pattern-Dependent

Relatively High-Level
**Bean Introspection**

- **Introspector**
  - `BeanInfo getBeanInfo(Class c)`

- **BeanInfo**
  - `PropertyDescriptor[] getPropertyDescriptors()`

- **PropertyDescriptor**
  - `Method getReadMethod()`
  - `Method getWriteMethod()`

- **Class**
  - `getMethods()`
Better Bean Introspection Interface

- **Introspector**
  - `BeanInfo getBeanInfo(Class c)`

- **BeanInfo**
  - `PropertyDescriptor[] getPropertyDescriptors()`

- **PropertyDescriptor**
  - `Object get(Object parent);`
  - `set(Object parent, Object component)`
List Introspection Provided by Oeall

ReflectionUtility

- static boolean isList(Class c)
- static int size(Object list)
- Object get (Object list, int index)
- void set(Object list, int index, Object component)

Why no isBean()?

Every object is a bean with 0 or more properties
## Language Support for Library Support

<table>
<thead>
<tr>
<th>Language support for libraries can be used to manipulate logical/physical structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Libraries needed as new patterns emerge</td>
</tr>
<tr>
<td>Libraries can change language support for serialization of physical/logical structures</td>
</tr>
</tbody>
</table>
PHYSICAL VS. LOGICAL COMPONENTS (REVIEW)

1. Handle Header (Node, ByteSequence)
2. ∀ component, C
   - Handle Component Header (C, ByteSequence)
3. Visit(Node, ByteSequence)
4. Visit(C, ByteSequence)
5. Atomic (C)?
6. Handle Atomic (C, ByteSequence)
**Comparison?**

- **Physical Structure:** Memory representation
- **Logical Structure:** Interface + Pattern
### Physical vs. Logical (Automation)

<table>
<thead>
<tr>
<th>Any object can be physically serialized automatically</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not all objects follow Bean Pattern</td>
</tr>
<tr>
<td>List pattern non standard</td>
</tr>
<tr>
<td>Not all communicated objects follow List and Bean pattern</td>
</tr>
<tr>
<td>In practice, Bean, List, and Hashmap patterns should suffice</td>
</tr>
</tbody>
</table>
In general physical state larger than exported state

Accommodates growth an in BMI List
public class AnObjectHistory<ElementType> implements ObjectHistory<ElementType>
{
    transient public final int MAX_SIZE = 50;
    transient Object[] contents = new Object[MAX_SIZE];
    int size = 0;
    public int size() { return size; }
    public ElementType get(int index) {
        return (ElementType) contents[index];
    }
    boolean isFull() { return size == MAX_SIZE; }
    public void add(ElementType element) {
        if (isFull())
            System.out.println("Adding item to a full history");
        else {
            contents[size] = element;
            size++;
        }
    }
}
Often physical serialization overridden to serialize physical structures

```java
public void writeObject(ObjectOutputStream stream) {
    try {
        stream.defaultWriteObject();
        for (int i = 0; i < size; i++)
            stream.writeObject(contents[i]);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void readObject(ObjectInputStream stream) {
    try {
        stream.defaultReadObject();
        contents = new Object[MAX_SIZE];
        for (int i = 0; i < size; i++)
            contents[i] = stream.readObject();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```
**Physical vs. Logical (Heterogeneity)**

Different physical structures can imply same logical structure

Can translate into object of different class

Different logical structures can imply same physical structure but received object not semantically equivalent

Semantically equivalence $\rightarrow$ same logical structure

```java
registerDeserializingClass(SerializedClass, DeserializedClass)
```

```java
registerDeserializingClass(Vector.class, ArrayList.class)
```
COMMUNICATING. PHYSICAL VS. LOGICAL STRUCTURE

AnotherIntHistory

size

int 1

contents

ArrayList

???

Always serializable

Not Isomorphic

AnIntHistory

size

int 1

contents

int []

int 1

int 3

int 0

IntHistory

[0]

int 3

Smaller Size

IntHistory

[0]

Can Deserialize into instance of different class based on some mapping

int 3

Always serializable

IntHistory

[0]

int 3
**Physical vs. Logical Infrastructure Supported**

**Physical Structure**
- Mesa, Cedar, Ada
- Python
- Java and .NET
- GIPC

**Logical Structure**
- Mesa, Cedar, Ada
- CORBA
- XML (Java and .NET) Serialization
- GIPC

In (strongly typed) conventional language logical = physical structure

Performance problems addressed by extensibility

Heterogeneity problems addressed by extendible XML Bean Serialization
Algorithm

Handle Header (Node, ByteString)

∀ component, C

Handle Component Header (C, ByteString)

Visit(Node, ByteString)

Visit(C, ByteString)

Atomic (C)?

Handle Atomic (C, ByteString)
BMISpreadsheet bmi =
\textbf{new} ABMISpreadsheet()

\begin{itemize}
  \item Works for unnamed components
  \item Both ordered and unordered collections (Set, List)
  \item Works when order of named fields is fixed
  \item Same language and compiler
  \item or Alphabetical sorting
  \item Full deserialization
  \item Known destination type
\end{itemize}
**Sending Component Names**

ABMISpreadsheet

- **height**
  - double 1.77

- **weight**
  - double 75.0

| "Height" | 1.77 | "Weight" | 75.0 |

Object

- C¹ Name & Value Serialization
- C² Name & Value Serialization
- C³ Name & Value Serialization
- ...
**Duplicated Names**

- AnExtendedBMISpreadsheet
  - height: double 1.77
  - weight: double 77.0
double 75.0

- ABMISpreadsheet
  - “Height” 1.77
- AnExtendedBMISpreadsheet
  - “Weight” 75.0

- Class $T^1$ Serialization
- Class $T^1$ Declared Components
- Class $T^2$ Serialization
- Class $T^2$ Declared Components

Can have multiple components with same name
- public variable and property, variable in different classes

Declared Components:
- double 1.77
- double 75.0
- double 77.0

Serialization:
- ABMISpreadsheet
- “Height” 1.77
- “Weight” 75.0
ALGORITHM

Handle Header (Node, ByteSequence)

∀ component, C

Handle Component Header (C, ByteSequence)

Visit(Node, ByteSequence)

Visit(C, ByteSequence)

Atomic (C)?

Handle Atomic (C, ByteSequence)

Send type names of components?
COMPOSING (POLYMORPHIC) OBJECT VALUES

```java
Object[] objects = {“hello”, new ABMISpreadsheet()}
```

Polymorphic Object Component

Other Component Serialization

Class Serialization

(Name & Value Serialization)
Basic Tree Algorithm

- Handle Header (Node, ByteSequence)
- ∀component, C
- Handle Component Header (C, ByteSequence)
- Visit(Node, ByteSequence)
- Visit(C, ByteSequence)
- Atomic (C)?
- Handle Atomic (C, ByteSequence)
- Non tree structures?
Visited Node and Types

Handle Header (Node, ByteSequence)

Visited (Node, Visited)?

∀ component, C

Handle Component Header (C, ByteSequence)

Handle Visited (C, ByteSequence, Visited)

Visit (C, ByteSequence, Visited)

Atomic (C)?

Handle Atomic (C, ByteSequence)

Implementation?
Object[] objects = new Object[2];
objects[0] = “hello”;
objects[1] = objects[0];
objects[2] = objects;

Algorithm keeps track of visited nodes
Visit number: number of first visits before a node first visited in recursive algorithm.
Assume corresponding nodes visited in the order in serialization and deserialization
Serialization serializes revisited node value by putting visit number of the node
Deserialization converts visit number to already constructed node
Node numbers are internal pointers within serialized representation
Repeated Types

Object[] objects = new Object[2];
objects[0] = “hello”;
objects[1] = “ca va”;

Supported in Java

What if all elements happen to be String?

Object[].class | String.class | “hello” | TypeRef | 1 | “ca va”
MESSAGE POINTERS

- Value (and type) pointer
- Type pointer
- Point to previously visited value and type
- Point to previously visited value type
Repeated Types

Object[] objects = new Object[2];
objects[0] = “hello”;
objects[1] = “ca va”;

What if all elements happen to be String?
@StringElement
Object[] objects = new Object[2];
objects[0] = “hello”;
objects[1] = “ca va”;
MESSAGE POINTERS

Value (and type) pointer

Point to previously visited value and type

Type pointer

Point to previously visited value type
**INDEPENDENT SERIALIZATIONS**

No Inter Message Compression

- String.class
- “hello”
- String.class
- “ca va”

Inter Message with Local Addressing and Headers

- Msg 0
- String.class
- “hello”
- Msg 1
- Type Ref
- 0.0
- “ca va”

Inter message with Global (type) Addressing

- String.class
- “hello”
- Type Ref
- 0
- “ca va”

Inter message compression reduces generality and correctness of serialization scheme
**INTER-MESSAGE COMPRESSION**

```java
ObjectOutputStream socketOut = new ObjectOutputStream(socket.getOutputStream());
Object[] objects = new Object[1];
objects[0] = "hello";
socketOut.writeObject(objects);
objects[0] = "ca va";
socketOut.writeObject(object);
```

**objects[0]**  Reference | 0

Stream could be re created or reset to release cache
MULTICAST AND SERIALIZATION
Could serialize separately for each destination

Serialization is expensive
COMMON Serializer

Assuming single serializer object for all receivers

Interferes with compression when multicast is not broadcast
MULTICAST VS. COMPRESSION

Assuming single serializer object for all receivers

Stream is corrupted when assumption breaks

Relayer
**Serialization Communication Implemented by Who**

- Infrastructure (Language, Library)
- Programmer (of Communicated Objects)
- Both (Programmer code called by Infrastructure)

**Reasons for overriding?**
EXTENSIBILITY REASONS

Some parts of objects cannot be serialized

Some parts of objects do not need to be serialized and need to be derived

May want a more efficient serialization scheme (sp. for physical structures)

As new patterns emerge, new (de) serializers needed

May need to determine which equivalent type to deserialize an object to
public class AnotherBMISpreadsheet implements BMISpreadsheet{
    double height = 1.77;
    double weight = 75;
    transient double bmi = calculateBMI();

    public double getHeight() { return height; }
    public void setHeight(double newHeight) {
        height = newHeight;
        bmi = calculateBMI();
    }

    public double getWeight() { return weight; }
    public void setWeight(double newWeight) {
        weight = newWeight;
        bmi = calculateBMI();
    }

    public double getBMI() { return bmi; }
    double calculateBMI() {
        return weight/(height*height);
    }
}"
public class AnotherBMISpreadsheet implements BMISpreadsheet{
    double height = 1.77;
    double weight = 75;
    double bmi = calculateBMI();
    public double getHeight() { return height; }
    public void setHeight(double newHeight) {
        height = newHeight;
        bmi = calculateBMI();
    }

    public double getWeight() { return weight; }
    public void setWeight(double newWeight) {
        weight = newWeight;
        bmi = calculateBMI();
    }

    @Transient
    public double getBMI() { return bmi; }
    double calculateBMI() {
        return weight/(height*height);
    }
}

Logical component returned by annotated method is transient
**Without Transients**

- Handle Header (Node, ByteSequence)
- Visit (Node, ByteSequence, Visited)

  - Visited (Node, Visited)?
  - ∀ component, C
    - Handle Component Header (C, ByteSequence)
    - Atomic (C)?
      - Handle Atomic (C, ByteSequence)
  - Handle Visited (C, ByteSequence, Visited)
  - Transients?

- Visit (C, ByteSequence, Visited)
WITHOUT TRANSIENTS

Handle Header (Node, ByteSequence)

Visited (Node, Visited)?

∀ non-transient component, C

Handle Component Header (C, ByteSequence)

Handle Visited (C, ByteSequence, Visited)

Visit(C, ByteSequence, Visited)

Atomic (C)?

Handle Atomic (C, ByteSequence)
**Extensibility Reasons**

- Some parts of objects cannot be serialized
- Some parts of objects do not need to be serialized and need to be derived
- May want a more efficient serialization scheme (esp. for physical structures)
- As new patterns emerge, new (de)serializers needed
- May need to determine which equivalent type to deserialize an object to

Custom (de)serializers?
BASIC ALGORITHM

Handle Header (Node, ByteSequence)

Visited (Node, Visited)?

∀ (non transient) component, C

Handle Component Header (C, ByteSequence)

Atomic (C)?

Visit(C, ByteSequence, Visited)

Handle Visited (C, ByteSequence, Visited)

Extensibility?

Visit(Node, ByteSequence, Visited)

Handle Atomic (C, ByteSequence)
**Generic vs. Type Visitor**

Handle Header (Node, ByteSequence)

Visited (Node, Visited)?

TypeVisit ← FindTypeVisitor(Node)

TypeVisit(Node, ByteSequence, Visited)

GenericVisit(Node, ByteSequence, Visited)

Handle Visited (C, ByteSequence, Visited)

FindTypeVisitor can return default or custom visitor
∀ non transient component, C

Handle Component Header (C, ByteSequence)

GenericVisit(C, ByteSequence, Visited)

Atomic (C)?

Handle Atomic (C, ByteSequence)

Call the top serializer
Typical Custom Type Visitor

∀ non transient custom component, C

Custom Handle Component Header (C, ByteSequence)

GenericVisit(C, ByteSequence, Visited)

Atomic (C)?

Handle Atomic (C, ByteSequence)
Generic vs. Type Visitor

Handle Header (Node, ByteSequence)

Visited (Node, Visited)?

TypeVisit ← FindTypeVisitor(Node)

TypeVisit(Node, ByteSequence, Visited)

GenericVisit(Node, ByteSequence, Visited)

Handle Visited (C, ByteSequence, Visited)
SPECIFYING CUSTOM (DE)SERIALIZERS IN JAVA

TypeVisit $\leftarrow$ FindTypeVisitor(Node)

(De)Serializer Class Implements Custom Code
ObjectOutputStream socketOut = new ObjectOutputStream(socket.getOutputStream());
socketOut.writeObject(3);
ObjectInputStream socketIn = ObjectInputStream(socket.getInputStream());
Integer readVal = (Integer) socketIn.readObject();
**Serializable Pattern based Functions**

Serializable

- `private readObject(ObjectInputStream s)`
- `private writeObject(ObjectOutputStream s)`

These methods called if they exist

Also used to initialize transients structures

ObjectInputStream

- `defaultReadObject()`

ObjectOutputStream

- `defaultWriteObject()`

Default methods (de)serialize non transients
private void readObject(ObjectInputStream stream) {
    try {
        stream.defaultReadObject();
        initSerializedObject();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
**EXTERNALIZABLE INTERFACE BASED FUNCTIONS**

- **Externalizable**
  - `readExternal (ObjectIn )`
  - `writeExternal(ObjectOut)`

These methods must exist if interface implemented

- **ObjectIn**
  - `Object readObject(Atomic)(Object (Atomic))`

- **ObjectOut**
  - `writeObject(Atomic)(Object(Atomic))`

No default read and write of Object
Both public interface methods must be implemented and no default read or write methods
### Serializable vs. Externalizable

#### Private Custom Methods
- Can make mistake in method signature
- Can invoke default methods
- Can only initialize transients
- Sends field names, field types and declaring super classes
- Less efficient

#### Public Custom Methods
- Must implement both methods
- No default methods
- Serialize both transient and non transients
- Sends only values explicitly written by programmer
- Cannot read data without class at receiver’s site

#### Alternatives?
Extensibility Reasons

- Some parts of objects cannot be serialized
- Some parts of objects do not need to be serialized and need to be derived
- May want a more efficient serialization scheme (esp. for physical structures)
- As new patterns emerge, new (de) serializers needed
- May need to determine which equivalent type to deserialize an object to

Alternatives?
SPECIFYING CUSTOM (DE)SERIALIZATION

TypeVisit ← FindTypeVisitor(Node)

(De)Serialized Class Implements Custom Code for Overriding Serialization and Initialization
SPECIFYING CUSTOM SERIALIZERS

TypeVisit ←
FindTypeVisitor(Node)

(De)Serialized Class Implements Custom Code for Overriding Serialization and Initialization

Special pattern for initializing serialized method
public class AnotherBMISpreadsheet implements BMISpreadsheet{
    double height = 1.77;
    double weight = 75;
    double bmi = calculateBMI();
    public double getHeight() {return height;}
    public double getWeight() {
        return weight;
    }
    public void setHeight(double newHeight) {
        height = newHeight;
        bmi = calculateBMI();
    }
    public void setWeight(double newWeight) {
        weight = newWeight;
        bmi = calculateBMI();
    }
    public double getBMI() {
        return bmi;
    }
    double calculateBMI() {
        return weight / (height * height);
    }
    public void initSerializedObject() {
        bmi = getBMI();
    }
}
Some parts of objects cannot be serialized

Some parts of objects do not need to be serialized and need to be derived

May want a more efficient serialization scheme (sp. for physical structures)

As new patterns emerge, new (de) serializers needed

May need to determine which equivalent type to deserialize an object to
CUSTOM SERIALIZATION

TypeVisit \leftarrow \text{FindTypeVisitor}(\text{Node})

\text{(D)e}Serialized Class Implements Custom Code for Overriding Serialization and Initialization

Special pattern for initializing serialized method

External Custom Serializer

Registry
## Custom External Serializer API

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>registerSerializer</code></td>
<td>Registers a custom serializer for a given type and pattern.</td>
</tr>
</tbody>
</table>

- `registerSerializer(Integer.class, integerSerializer);`
- `registerSerializer(HashSet.class, collectionSerializer);`
- `registerSerializer(BEAN_PATTERN, listSerializer);`
- `registerSerializer(LIST_PATTERN, collectionSerializer);`
COMPARISON?

TypeVisit $\leftarrow$
FindTypeVisitor(Node)

(De)Serialized Class Implements Custom Code for Overriding Serialization and Initialization

Special pattern for initializing serialized method

External Custom Serializer

Registry
COUPLING OF TRANSIENTS INITIALIZATION AND EXTENSIBILITY

- Extensibility and initialization of transients is coupled
- Often they are done together
- Only one set of abstraction learnt
COUPLING OF INITIALIZATION AND EXTENSIBILITY

Serializable: App must be aware of and call defaultReadObject()

```java
private void readObject(ObjectInputStream stream) {
    try {
        stream.defaultReadObject();
        initSerializedObject();
    } catch (Exception e) { e.printStackTrace(); }
}
```

Both: App must know about input stream and deserialization exception handling

```java
public void readExternal(ObjectInput in) {
    try {
        height = in.readDouble();
        weight = in.readDouble();
    } catch (Exception e) { e.printStackTrace(); }
}
```

Externalizable: App must do complete serialization and deserialization

```java
public void writeExternal(ObjectOutput out) {
    try {
        out.writeDouble(height);
        out.writeDouble(weight);
    } catch (Exception e) { e.printStackTrace(); }
}
```

Separating interface (method signatures) for extensibility and transient initialization does not have this problem
COUPLING OF HANDLING AND EXTENDED SERIALIZATION

Serialization extension must be done by the class being serialized.

Allows extension to easily and efficiently access physical structure (without reflection).
INTERNAL SERIALIZER

Cannot add serializer for class without source code

Must inherit and change references and not reuse code
COUPLING OF HANDLING AND EXTENDED SERIALIZATION

Cannot have multiple serializers for different contexts

Text vs Binary, Logical vs. Physical, Mobile vs. Desktop, Single Language vs. Multiple Language
Generic vs. Type Visitor

- Handle Header (Node, ByteSequence)
- GenericVisit(Node, ByteSequence, Visited)
- Change generic, top level visitor?
- Handle Visited (C, ByteSequence, Visited)
- Visited (Node, Visited)?
- TypeVisit \leftarrow FindTypeVisitor(Node)
- TypeVisit(Node, ByteSequence, Visited)
Java

ObjectInputStream (InputStream i)

Object readObject()

classes to which RMI and other users are bound

ObjectOutputStream (OutputStream o)

writeObject(Object o)

GIPC

Object objectFromInputBuffer (ByteBuffer)

Interface, whose instances are instantiated through factories

ByteBuffer outputBufferFromObject (Object)
GIPC Serializer Factory and Selector

- Serializer Factory
  - Serializer createSerializer()

- Serializer Selector
  - static SerializerFactory serializerFactory
  - static Serializer createSerializer()
**GIPC Serializer**

ASimple Serializer

ASingleStream Serializer

ACustom Serializer

implements

Object objectFromInputBuffer(ByteBuffer)

ByteBuffer outputBufferFromObject(Object)
Use serializer to convert object to byte buffer

Use byte-buffer sender to send byte buffer

Use deserializer to deserialize byte buffer

Use byte buffer receiver to receive byte buffer

Serializer may reuse buffer for serialization

Send is not synchronous
**Output Buffer Management**

Create separate serializer for each send

- Does not allow sharing – defeats single buffer serialization

Some serializers create separate buffer for each send, wasted instantiation
**Serializer Pool**

- `Serializer` extends `SerializerPool`
- `objectFromInputBuffer(ByteBuffer)`
- `ByteBuffer outputBufferFromObject(Object)`
- `messageSent(String aRemoteEnd, ByteBuffer message, int sendId)`

- Serializer pool used instead of serializer
- Next free serializer from pool used to serialize
- Serializer returned to pool when message sent
Use serializer pool to convert object to buffer

Use byte buffer sender to send buffer

Use serializer to deserialize buffer

Use byte buffer receiver to receive buffer

Deserializer buffer management?

Byte buffer can be reused by reading thread after object copies made

If deserialization fails, bytebuffer returned, which must be copied by receiver
SUPPORTING MULTIPLE LANGUAGES

Object in Language 1  Object in Intermediate Description Language (IDL)  Object in Language 2

If languages are OO does IDL for Serialization have to be O-O?

Describing structures, OO would require decoding the structure of the IDL object
public class ABMISpreadsheet implements BMISpreadsheet {
    double height = 1.77;
    double weight = 75;
    public double getWeight() { return weight; }
    public void setWeight(double newWeight) {
        weight = newWeight;
    }
    public double getHeight() { return height; }
    public void setHeight(double newHeight) { height = newHeight; }
    public double getBMI() { return weight/(height*height); }
}

new ABMISpreadsheet()

typedef struct {
    double height;
    double weight;
} ABMISpreadsheet;

AMBMISpreadsheet myBMI = {1.77, 75}
XML Schema and Instance

```
<ABMISpreadsheet xmlns="urn:xmlns:25hoursaday-com:ABMISpreadsheet">
  <height>1.77</height>
  <weight>75.0</weight>
</ABMISpreadsheet>
```

Common schema for all Beans?
Representation based on Pattern-Specific, Class Independent Schema

<Object xmlns="urn:xmlns:25hoursaday-com:Object">
  <Object>
    <Property><Key> height </Key> <number>1.77</number></Property>
    <Property><Key> weight </Key> <number>75.0</number></Property>
  </Object>
</Object>

More verbose, structural rather than type equivalence
**Automatic vs. Manual Translation**

- **Object in Language 1**
- **Object in Intermediate Description Language (IDL)**
- **Object in Language 2**

---

**Automatic translation for Beans**

- Schema generated automatically
- Converter generated automatically

---

**Constrains patterns programmers use**

- No need to create and update schema
- No need to translate between language and schema representation
JSON: JAVASCRIPT BASED IDL

```json
{ "name" : "ABMISpreadsheet", "properties":
  { "height": { "type" : "number", "required" : true },
    "weight" : { "type" : "number", "required" : true }
  }
}

{ "height": 1.77, "weight" : 75.0 }
```
AUTOMATIC VS. MANUAL TRANSLATION (REVIEW)

Object in Language 1

Object in Intermediate Description Language (IDL)

Object in Language 2

C, XML, JSON, ....

Need to create and update schema

Need to translate between language and schema representation

Automatic translation for language independent patterns such as Bean
SERIALIZATION AND STRONG TYPING

```c
char *s;
```

Physical structure communicated and reconstructed?
**Array Interpretation**

```c
char *s;
s = {'a', 'b', 'c'}
```
**STRING INTERPRETATION**

```c
char *s;
s = "abc";
```

```
'a'
'b'
'c'
'
```
CHAR POINTER INTERPRETATION

```
s = &'a';
char *s;
s = &'a';
```
Like subclasses, union describes alternatives

Each alternative is a way of interpreting memory

union AnIntOrChars{
    int s;
    char c[4];
} intOrChars

intOrChars

0
0
0
0
65
**INT INTERPRETATION**

```c
union AnIntOrChars{
   int s;
   char c[4];
} intOrChars

intOrChars.s = 65
```

Diagram:
- `union AnIntOrChars{
   int s;
   char c[4];
} intOrChars`
- `intOrChars.s = 65`
- `int 65`
**Char Array Interpretation**

```c
union AnIntOrChars{
    int s;
    char c[4];
} intOrChars

intOrChars.s [0]= ""
```

<table>
<thead>
<tr>
<th>intOrChars</th>
<th>AnIntOrChars</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>char &quot;&quot;</td>
</tr>
<tr>
<td>0</td>
<td>char &quot;&quot;</td>
</tr>
<tr>
<td>0</td>
<td>char &quot;&quot;</td>
</tr>
<tr>
<td>0</td>
<td>char 'A'</td>
</tr>
<tr>
<td>65</td>
<td>char &quot;&quot;</td>
</tr>
</tbody>
</table>
**Strong Typing and Serialization**

In weakly typed languages type of a variable is ambiguous.

Programmers disambiguate based on their application-specific knowledge.

Serialization depends on type of variable: same memory can be serialized differently based on its type.

Serialization requires unambiguous typing.

Give up on C?
**STANDARD SOLUTION: IDL**

- Object in Language 1
- Object in Intermediate Description Language (IDL)
- Object in Language 2

Need to create and update schema

Need to translate between language and schema representation
Suite Directive-based Serialization

```c
char *s;

/*oc
DiscriminatedUnion AnIntOrCharsStruct
*/
struct AnIntOrCharsStruct{
    enum {isShort, isChars} tag;
    union {
        int s;
        char c[4];
    } intOrChars;
} intOrCharsStruct;
```

Interpret as string (default)

Use enum field storing one of N choices preceding union with N alternatives as discriminant, with choice I implying alternative I.

Special comments used as directives for serialization

Serialization directive is a general idea supported in later systems through other language-supported constructs.
If the type of an object is unambiguous, is it serializable?

Conceptual and implementation issues
**Unknown Type**

Sender

Value: T

Receiver

Receiver does not know about the type of sent values

Application may be type-independent (Logger, Monitor, Registry, Realeyer)

Type may have evolved

How to support receipt of value of unknown type
No Handling of Received Unknown Value

Sender
Value: T

Receiver
Value: T

Storage
Value: T

Forwarder
Value: T

If no Handling, can simply forward or store received representation.

Need a way to get serialized buffer for value

Not easy in Java
## What Handling to do with Unknown Type

<table>
<thead>
<tr>
<th>Can invoke some method of known super type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applet, Thread start method</td>
</tr>
<tr>
<td>Relies on dynamic dispatch in virtual methods (Java)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Can invoke some method of known signature using reflection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invoke main method, getter, setter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Handle method invocation request from requester that knows type and possibly sent the object</th>
</tr>
</thead>
<tbody>
<tr>
<td>A client loads ABMISpreadsheet and same and other clients make remote calls in it</td>
</tr>
</tbody>
</table>
**Message Handling: Same Language**

Applets are loaded dynamically

There is a way in Java to load classes from the network
### Dynamic Loading

- **URLClassLoader**
  - `URLClassLoader(URL[] aSearchPath)`
  - `Class loadClass(String aFullClassName)`

- **URL**
  - `URL(String aURLString)`
```java
public static Class loadClass(
    String aURLOfClassesFolder, String aFullClassName) {
    try {
        URL[] urlSearchPath = { new URL(aURLOfClassesFolder) };
        URLClassLoader loader =
            new URLClassLoader(urlSearchPath);
        return loader.loadClass(aFullClassName);
    }
    catch (MalformedURLException e) {
        e.printStackTrace();
        return null;
    }
    catch (ClassNotFoundException e) {
        e.printStackTrace();
        return null;
    }
}

loadClass("www.cs.unc.edu/~dewan/comp734/gipc/classes", "bmi.ABMISpreadsheet")
```
**Message Handling: Same Language**

In Java, URL for location to load class can be sent to receiver.

- **URL returned by** `RMIClassLoader.getClassAnnotation(class)`
- **Must return non null in Web Servers**

It would be nice if classes were communicated without requiring URLs.
Message Handling: Different Language

Could convert to special data structure of known type to which graph is translated using sent schema:

- e.g. XML Node, Generic Dynamic Bean with property names as arguments to setters and getters
- e.g. New ABMISpreadsheet with setters/getters
- Would lose semantics such as getBMI()

Cannot do app-specific computations unless we have an O-O IDL
SERIALIZER AND COMMUNICATION

Sender <Object> (Object Port)

Send Forwarder <Object, Object>

Serializer <Object, ByteBuffer>

Sender<ByteBuffer> (Buffer Skeleton)

Receive Notifier<Object>

ReceiveForwarder <Object, Object>

Deserializer <ByteBuffer, Object>

Receive Listener <ByteBuffer> (Buffer Skeleton)
Serialize and Multicast Alternatives

- Typed Data Communication
  - Multicast and serialize
    - (Expensive!) Serialization done multiple times
  - Serialize and multicast
    - (Expensive!) Serialization done once
Object Communication vs. RPC, Serialization vs. Marshaling of Values

- Object Reference Communication
- Remote Procedure Call
- Object Copy Communication
- Byte Communication

(De)Serialization

(Un)Marshaling
Byte Buffer?

Handle Header (Node, ByteSequence)

Visited (Node, ByteSequence, Visited)?

∀component, C

Handle Component Header (C, ByteSequence)

Visit(C, ByteSequence, Visited)

Atomic (C)?

Handle Atomic (C, ByteSequence)

Java Stream vs. Java ByteBuffer?

Visit(Node, ByteSequence, Visited)

Handle Visited (C, ByteSequence, Visited)
**Stream vs. ByteBuffer**

In stream based serialization, conceptually one stream for all communication between two parties.

In byte buffer based serialization, conceptually each byte buffer is independent of the previous ones sent from that destination.

Simulating ByteBuffer from Stream?
**Stream vs. ByteBuffer API Examples**

**Stream**
- ObjectInputStream (InputStream i)
- Object readObject()

**ObjectInputStream**
- ObjectInputStream (InputStream i)
- Object readObject()

**ObjectOutputStream**
- ObjectOutputStream (OutputStream o)
- writeObject(Object o)

**ByteBuffer**
- objectFromInputBuffer(ByteBuffer)
- outputBufferFromObject(Object)

**Serializer**
- Object objectFromInputBuffer(ByteBuffer)
- ByteBuffer outputBufferFromObject(Object)
public ByteBuffer outputBufferFromObjects(Object[] objects) {
    try {
        SeekableByteArrayOutputStream byteArrayOutputStream =
            new SeekableByteArrayOutputStream();
        ObjectOutputStream objectOutputStream =
            new ObjectOutputStream(byteArrayOutputStream);
        for (int i = 0; i < objects.length; i++) {
            objectOutputStream.writeObject(objects[i]);
        }
        objectOutputStream.flush();
        ByteBuffer retVal = ByteBuffer.wrap(byteArrayOutputStream.getBuffer(), 0, byteArrayOutputStream.size());
        return retVal;
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
```java
public List<Object> objectsFromInputBuffer(ByteBuffer inputBuffer) {
    try {
        ObjectInputStream objectInputStream = new ObjectInputStream(
            new ByteArrayInputStream(
                inputBuffer.array(),
                inputBuffer.position(),
                inputBuffer.remaining()));
        List<Object> retVal = new ArrayList();
        while (true) {
            try {
                Object readObject = objectInputStream.readObject();
                retVal.add(readObject);
            } catch (EOFException eof) {
                break;
            }
        }
        return retVal;
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
```
**Cost of Serializing**

- Each serialization creates a new byte array and streams around it.
- Could reuse the same stream for multiple serializations.
- Java Serializer adds 4 synchronization bytes at start and Deserializer removes them.
- Plus Java does inter message compression.
- Scheme reduces correctness and generality (same object being sent multiple times and multicast).
- ObjectOutputStream reset does not seem to work.