Interoperability in Programming Languages

Todd Malone

Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA

28 April 2014
Senior Seminar

What is interop?

- Interoperability: The ability for a system to use parts from another system.
- Often shortened to interop.
- In programming languages: The ability of a language to call on code from another language.

Why is interop important?

Developer time and effort

- Existing and working code is easier to use as-is.
- Third-party systems: source code is unavailable
- Legacy systems: extensive or little-understood code base.

Language Strength:

- Explicit memory access (C)
- Parallel or distributed systems (Clojure, Erlang)
- Statistics (R)

Outline

Outline

Differences in type systems

- Languages represent data in different ways
- Statically-typed languages assign types as soon as data is collected.
- Dynamically-typed languages only deal with types when evaluating data.
-

```
Class Person
  string name = "Cliff"
  date dateOfBirth = 4/16/1978
  int height = 74
  double weight = 212
end
```

```
Class Person
  var name = "Cliff"
  var dateOfBirth = 4/16/1978
  var height = 74
  var weight = 212
end
```

Mismatched structures

- Untyped lists can contain different types,
- Strongly typed lists can only contain the type given by the list.

```
[23, v, "hello", True]
```
An untyped list

```
[1, 53, 13, 100]
```
a typed list

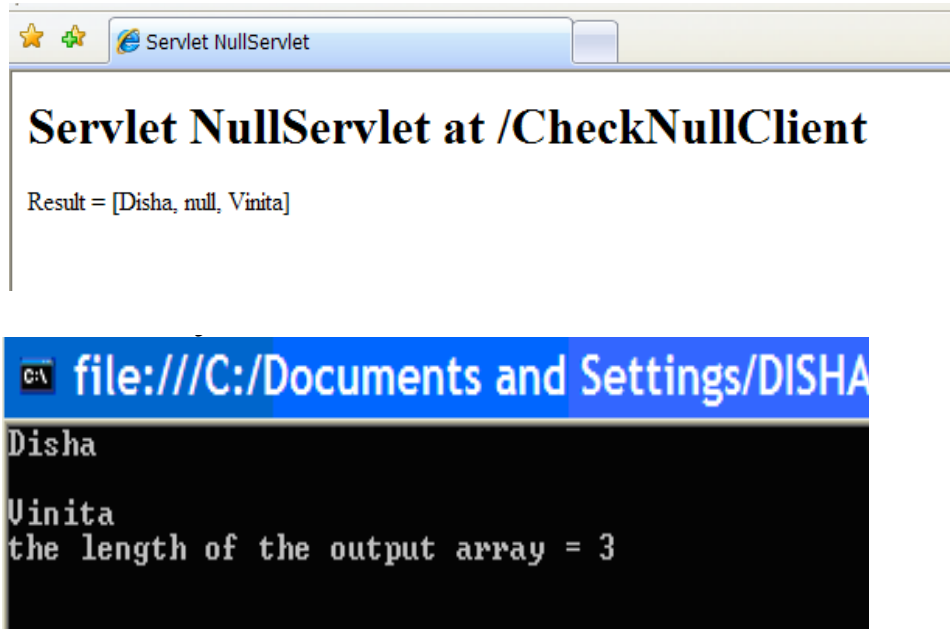
```
Object[] = [?, ?, ?, ?]
```
A Java list of Objects

- A data structure in one language may be absent in another
- Maps are common data structures, but absent in C.

```
{:name "Cliff", :age 32}
```
A map

Handling data

- Languages act on data in different ways.
- Handling NULL or NIL objects.
- Decimal precision: Java returns 12.999999, .NET returns 13



images based on Shetty and Vadivel[1]

Outline

Metadata and type conversion

Metadata: Data about data
or: Information beyond what the data itself can convey

```
(def mylist [1, 2, 3, 4])
(with-meta mylist {:length 4, :type Integer})
```

In Clojure:

- lists are untyped; can contain entries of different types.
- metadata, added as above, is all user-controlled.

Metadata and type conversion

no metadata
Clojure: `[1, 2, 3, 4]`
(down arrow)
Java: `[?, ?, ?, ?]`

metadata: `{:type Integer}`
Clojure `[1, 2, 3, 4]`
(down arrow)
Java `[1, 2, 3, 4]`

Metadata and standards

Metadata: `{:type int}`
[1, 2, 3, 4] -> Metadata: `{:type Integer}`
[?, ?, ?, ?]

Metadata: `{:type Integer}` -> Metadata: `{:type Integer}`
[1, 2, 3, 4] [1, 2, 3, 4]

The importance of standards

Standards are meant to ensure:

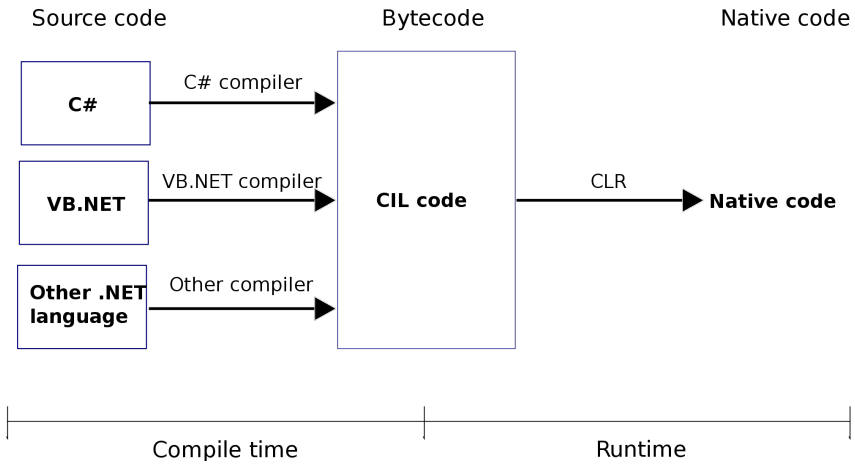
- Agreement on what metadata is being used, and how.
- All involved parties know how data will be represented.
- Future parties will know how data is represented.
- In general, that correct communication happens.

```
{
 "name": "Person",
 "properties": {
   "name": {
     "type": "string"
   },
   "birthdate": {
     "type": "date"
   },
   "height": {
     "type": "number"
   },
   "weight": {
     "type": "number"
   }
 }
}
```

Outline

Virtual machines

- Virtual Machines (VMs) are a runtime environment for a program
- High-level languages compile to an intermediate language
- Intermediate language: Java bytecode or Common Intermediate Language



Wikipedia
`https://en.wikipedia.org/wiki/Common_Language_Runtime`

High-level vs Bytecode

```
public class Fib{

public int fibonacci(int n) {
    if(n == 0){
        return 0;
    }else if(n == 1){
      return 1;
    }else{
      return fibonacci(n - 1) + fibonacci(n - 2);
    }
}

}
```

```
public class Fib {
  public Fib();
    Code:
       0: aload_0
       1: invokespecial #1
       4: return

  public int fibonacci(int)
    Code:
       0: iload_1
       1: ifne          6
       4: iconst_0
       5: ireturn
       6: iload_1
       7: iconst_1
       8: if_icmpne     13
      11: iconst_1
      12: ireturn
      13: aload_0
      14: iload_1
      15: iconst_1
      16: isub
      17: invokevirtual #2
      20: aload_0
      21: iload_1
      22: iconst_2
      23: isub
      24: invokevirtual #2
```

Interoperability with virtual machines

- Usually some overheard associated with calling other languages.
- Overhead can be lessened when all languages are on one VM.
- High-level languages can have conventions to call other high-level languages on the same VM.
- Common language ensures common syntax and behavior.

A Java method of object cliff:

```
cliff.getAge();
```

Clojure calling Java:

```
(.  getAge cliff)
```

JRuby calling Java:

```
require 'java'
cliff.getAge()
```

Markup languages

- Markup languages are a way of modeling data.
- XML and JSON can model data like objects.
- Markup languages are independent of programming languages.

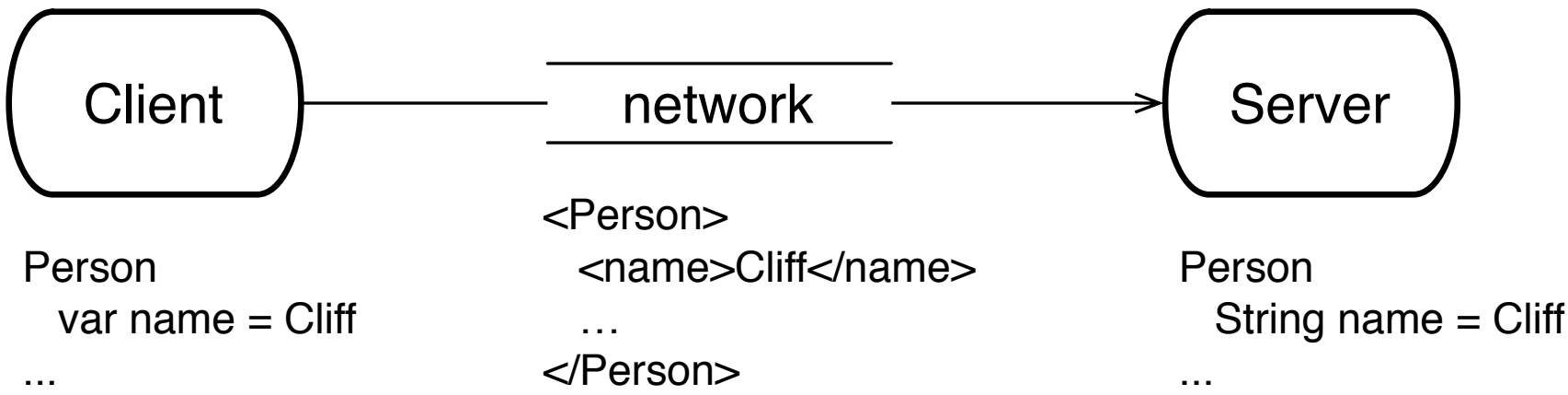
```
<Person>
  <name> Cliff </name>
  <birthdate> 4/16/1978 </birthdate>
  <height> 74 </height>
  <weight> 212 </weight>
</Person>
```

```
{
 "name": "Cliff",
 "birthdate": "4/16/1978",
 "height": "74",
 "weight": "212";
}
```

Interoperability with markup languages

- Act as Metadata.
- Schema provide both standardization and additional metadata.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="Person">
   <xs:complexType>
     <xs:sequence>
       <xs:element name="name" type="xs:string" />
       <xs:element name="birthdate" type="xs:date" />
       <xs:element name="height" type="xs:double" />
       <xs:element name="weight" type="xs:double" />
     </xs:sequence>
   </xs:complexType>
 </xs:element>
</xs:schema>
```


Client
network
Server
Person
var name = Cliff
...
<Person>
<name>Cliff</name>
…
</Person>
Person
String name = Cliff
...

Outline

Conclusions

-
-
-
-
-

Thank you for listening!

Contact: malone153@morris.umn.edu

Questions?

References


D. S. V. Sujala D Shetty.
Interoperability issues seen in web services.
IJCSNS International Journal of Computer Science and Network Security, 9:160–169, August 2009.