Developing Ada Applications for the Java Platform with JGNAT

Gary Dismukes
Ada Core Technologies
dismukes@gnat.com

Franco Gasperoni
ACT Europe
gasperon@act-europe.fr
Summary

✔ JGNAT: the Ada environment for the JVM

- JGNAT Compiler
- Tools
- Interoperability
  - Mapping Java to Ada
  - jvm2ada
- JGNAT Example
Java Platform

Java Language

Java Virtual Machine

APIs and Libraries
Java Benefits

✔ Portability (write once, run anywhere)

✔ Built-in security
Increasing Application Reliability on the JVM

Java Language

Ada

Java Virtual Machine

APIs and Libraries
Why Ada on the JVM?

✔ Ada is a more reliable language
  - Java is safe but not necessarily reliable

✔ Ada reliability covers all software aspects
  - not just pointers or array bounds

✔ Ada is more portable than Java for certain application domains
  - e.g. real-time systems
✔ Some important features are missing in Java
  – unsigned types
  – fixed-point
  – scalar abstraction
  – generics
  – high-level concurrency constructs
  – ...

✔ Ada scales up with application size
  – Separate specification & implementation

✔ Ada was designed to be run efficiently on native hardware as well as VMs with GC
JGNAT
Ada Environment for the JVM
✔️ Java is Ada friendly

✔️ JVM is an excellent Ada platform

✔️ Java APIs are a fundamental component of the technology
What is JGNAT

Port of GNAT Professional
– to the Java platform

+ Number of Java specific tools

JGNAT generates regular JVM bytecode
JGNAT Generates Regular JVM Bytecode

✔ This means that ...

- JGNAT is 100% compatible with existing JVMs
- You can use regular JVM tools on class files produced by JGNAT (e.g. debugger)
- **Transparent access to the Java API**
- **Interoperability with Java (the language)**
Objective: Implement 100% of Ada 95

JGNAT compiler entirely written in Ada
  - Front End & Back End

Sources available under GPL license
JGNAT Summary

- GNAT-to-JVM Compiler
- Tools
- Interoperability
- Mapping Java to Ada
- JGNAT Example
GNAT to JVM Compiler
JGNAT Compiler

✔ Implements all the Ada library
  – Ada.Text_IO, ...

✔ Certain features have no impact on generated code
  – some representation clauses, ...

✔ Real Time Annex compliance if underlying JVM supports it
  – Thread priorities, ...
Very Natural Mapping between Ada 95 & JVM

<table>
<thead>
<tr>
<th>Ada</th>
<th>JVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>scalars</td>
<td>primitive types</td>
</tr>
<tr>
<td>arrays</td>
<td>arrays</td>
</tr>
<tr>
<td>records</td>
<td>final classes</td>
</tr>
<tr>
<td>tagged types</td>
<td>classes</td>
</tr>
<tr>
<td></td>
<td>classes (sometimes packages)</td>
</tr>
<tr>
<td>packages</td>
<td>methods</td>
</tr>
<tr>
<td>subprograms</td>
<td>constructors</td>
</tr>
<tr>
<td>default initializations</td>
<td>exceptions</td>
</tr>
</tbody>
</table>
The JVM makes it easy for ... 

- No memory layout needed
- Stack machine for computations
- OO model built in
- Exception model built in
- Protected record-like mechanism built-in
- Index checks for free
- Null-dereference checks for free
Extra work needed for

✓ Deep copy operations \((x := y)\)
✓ Access to subprograms
✓ Nested subprograms (static chains)
✓ Passing slices by copy makes life easier
Fortran, Java, C++, C, Ada, GCC, Front ends, Backend, target machine code, Unix, VMS, NT, OS/2, VxWorks, target independent.

Target description:
x86, PowerPC, 68K, MIPS, Sparc, Alpha, i960, IA-64, HP-PA, ...
The JGNAT Backend

Ada → GNAT → GIGI → GCC back-end → Machine code

Ada → GNAT → Jexpand → Code Gen. → Java bytecode
GNAT & Ada Tasking

Your code

High-level tasking services

Target independent low-level tasking API

Gnat runtime

Your Ada application

Linux

VxWorks

OS/2

Windows NT
JGNAT Tasking & JVM

- Ada task = Java thread
- Ada code with tasks can call Java code with threads & conversely!
- Lock/unlock map onto JVM monitor operations
- Task priorities map onto Java thread priorities
- Real time Ada semantics on JVMs with real time behavior
- GNULLI primitives map nicely into JAVA thread ops
Tools
Conventional GNAT Tools...

- jgnat, jgnatmake, jgnatbind, jgnatlink
  - Standard GNAT tools available

- jgnatxref, jgnatfind, gnathtml

- gnatprep
  - GNAT preprocessor

- gnatstub
  - Generate stubbed compilable bodies from specs

- gnatelim
  - Detect and eliminate unused subprograms on a complete partition

- ....
Some JGNAT Tools...

✔ jvmstrip
  - Strips all debugging info from .class files

✔ jvmlist
  - Displays class file contents
  - Embeds original source into class bytecode

✔ jvm2ada
  - produces an Ada spec from any class file
  - allows transparent use of any Java API from Ada
✓ jarmake
  - Given a `.class` file gathers in a JAR file all `.class` files transitively needed
  - Useful for stand-alone applications & applets
  - Can use to include images and other data in JAR

✓ Any tool that works at the bytecode level works with JGNAT
  - Debugger
JGNAT Debugger

☑ Any JVM debugger works with GNAT

☑ Debugger needs specialization for Ada 95 to
  - display data in Ada 95 format
  - understand Ada 95 syntax
JVM Debugging Technology

JNDI = JN Debugging Interface

Could be the same machine

JNI = Java Native Interface

JNI = JNI = Java Native Interface

your app
Interoperability
Java APIs are a fundamental component of the technology.
JGNAT provides full Ada - Java interoperability at the language & API level.
Full Java interoperability

- ✔ Access any Java services from Ada
- ✔ Use Ada services from Java
- ✔ Extend Java classes & Ada tagged types across languages
- ✔ Propagate & catch exceptions across languages

Mix & match Ada and Java as you see fit
Ada & Java Interfacing
Summary

- Tool for automated interfacing (jvm2ada)
- Mapping Java classes to Ada packages
- Pragma Import and Convention
- Handling circularities
- Implicit conversions
- Support for Java interfaces
Accessing the Java API with jvm2ada

✓ 100% Java-to-Ada API conversion tool
✓ Absolutely no manual intervention
✓ Works on EVERY Java library
✓ Distribute the tool, NOT Ada bindings
Automated Interfacing to Java
Using jvm2ada

✔ Using Java APIs from Ada should be easy
✔ All API functionality must be available
✔ jvm2ada generates Ada specs from
  - .class, .zip and .jar files
✔ No hand-editing of Ada required
✔ Can map ALL Java classes onto Ada packages
✔ Handles circularities in Java API classes
✔ Supports Java interfaces and constructors
Mapping Java Classes to Ada Packages

✔ Each Java class is represented by an Ada library package
✔ The Ada package contains a single tagged type with components and primitive operations
✔ Uses pragmas Import and Convention
✔ Java-specific pragmas for interfaces and constructors
✔ Java packages and nested classes use child units
Pragma Import and Convention

✔ Pragma Import (Java, `entity_name`, `[external_name]`)
  – On packages (JGNAT maps to a Java class)
  – On subprograms (JGNAT maps to a Java method)
  – On variables (JGNAT maps to a static field)

✔ Pragma Convention (Java, `entity_name`)
  – On a tagged type (JGNAT maps to a Java class)
  – On a subprogram (JGNAT uses Java conventions)
Direct Mappings

✔ Abstract Classes & Methods
✔ Static Methods and Static Fields
✔ Final Fields: constants
public class Foo {
    public int Field;
    public void Some_Op (int p) {...}

    static public int Var;
    static public int Func (Foo obj, Foo [] b) {...};
}

Classes
Using Foo’s services in Ada

with Java;
with Foo;   use Foo;
procedure Client is
  Obj : Foo.Foo.Ref := new_Foo;
  A    : Foo.Foo.Arr  := new Foo.Foo.Arr_Obj (2..9);
  X    : Java.int := Foo.Foo.Func (Obj, A);
begin
  Some_Op (Obj, X);
  Obj.Field := 22;
  Foo.Var := 3;
end Client;

public class Foo {
  public int Field;
  public void Some_Op (int p) {...}
  static public int Var;
  static public int Func (Foo obj, Foo [] b) {...};
Let’s look at the real foo.ads
package Foo is
  type Typ is new java.lang.Object.Typ with record
    Field : Java.int;
  end record;
  type Ref is access all Typ’Class;
  type Arr_Obj is array (Natural range <> ) of Ref;
  type Arr is access all Arr_Obj;

  function new_Foo (This : Ref := null) return Ref;
  procedure Some_Op (This : access Typ; P : Java.int);
  function Func (Obj : access Typ’Class; B : Arr) return Java.int;

  Var : Java.int;
end Foo;
public class Object {
    ...
    public String toString();
}

public final class String {
    ...
    public static String valueOf (Object obj);
}
with type java.lang.String.Ref is access;
package java.lang.Object is
    type Typ is tagged limited null record;
    function toString (This : access Typ) return java.lang.String.Ref;
...
end java.lang.Object;

with type java.lang.Object.Typ is tagged;
with java.lang.Object;
package java.lang.String is
    type Typ is new java.lang.Object.Typ with null record;
    type Ref is access all Typ'Class;
    function valueOf (Obj : access java.lang.Object.Typ'Class)
        return Ref;
end java.lang.String;
with Java.Lang.Object;
with Java.Lang.String;

procedure Client is
    Obj : java.lang.Object.Ref := ...;
    Str : java.lang.String.Ref := toString (Obj);
    ...

Implicit Conversions

✔ Problem: Java more weakly typed than Ada 95, making use of access types cumbersome

✔ Java:
  – Allows freely passing objects to parent operations
  – Null can be passed to noncontrolling arguments

✔ Ada 95:
  – Can have many access types (e.g., one per tagged type)
  – Named access type parameters do not allow implicit conversions
  – Access parameters allow implicit conversions, but restricted in use (null actual causes exception)
Possible Solutions

✔ Relax Ada rules for passing null to access parameters (for Java-convention subprograms)
  – Problem: Not clearly legitimate semantics

✔ Syntactic GNAT-specific language extensions
  – Special syntax for "open" access types that allow implicit conversions
  – Problem: Nonstandard extension

✔ Official language extension as part of with type
  – Implicit conversions allowed for all general access types (access all, access constant)
  – Problem: Might cause ambiguities (incompatibility)
Java-Specific Pragmas

✓ Java_Interface

✓ Java_Constructor
Java Interfaces

✔ Java interface types allow a nice form of multiple inheritance not supported by Ada 95

✔ A Java class may "implement" multiple interfaces, inheriting and overriding interface methods

✔ Ada tagged types can only extend from a single parent

✔ Java API classes depend on interfaces

✔ Generic mix-ins are a possible mechanism, but unwieldy
Using Java Interfaces from Ada

✔ A Java interface is mapped onto an abstract tagged type with abstract operations (pragma Java_Interface)

✔ The tagged type has a single access discriminant:

```ada
type Some_Interface (Self : access Java.Lang.Object'class) is
  new abstract Java.Lang.Object with null record;
pragma Java_Interface (Some_Interface);
```
 DialogResult

type Interface_User

  (I_Some_Interface : access Some_Interface'class;
   I_Other_Interface : access Other_Interface'class;
   ... etc.)
  is new Parent_Type with ...;
with Java.Lang.Object; use Java.Lang.Object;
package Fab is
  type Typ (Self : access Object.Typ'Class)
    is new abstract Object.Typ with null record;
  pragma Java_Interface (Typ);
  type Ref is access all Typ'Class;
procedure Proc (This : access Typ; Val : Float) is abstract;
function Func (This : access Typ) return Float is abstract;
end Fab;
with Java.Lang.Object; use Java.Lang.Object;
with Fab;
package Bar is
    type Typ (I_Fab : access Fab.Typ’Class)
        is new Java.Lang.Object.Typ with
    record
        Field : Integer;
    end record;
end record;
type Ref is access all Typ’Class;

procedure Proc (This : access Typ; Val : Float);
function Func (This : access Typ) return Float;
...

Conversions

with Java.Lang.Object; use Java.Lang.Object;
with Fab;
package Bar is
    type Typ (I_Fab : access Fab.Typ'Class)
        is new Java.Lang.Object.Typ with record
            Field : Integer;
        end record;
    type Ref is access all Typ'Class;

    procedure Proc (This : access Typ; Val : Float);
    function Func (This : access Typ) return Float;
end Bar;

package Fab is
    type Typ (Self : access Object.Typ'Class)
        is new abstract Object.Typ with null record;
    pragma Java_Interface (Typ);
    type Ref is access all Typ'Class;

    procedure Proc (This : access Typ; Val : Float)
    function Func (This : access Typ) return Float
end Fab;

X : Bar.Ref := new_Bar;
Y : Fab.Ref := X.I_Fab;  -- OK - conversion allowed, no checks.
Z : Bar.Ref := Bar.Ref (Y.Self);  -- At run time, check that
    -- Y points to an object in Bar.Obj'Class
public class C {
    public int field;
    public C () { field = 3; }
    public C (int i) { field = i; }
    public C (int i, int j) { this (i + j); }
}

with Java;
with Java.Lang.Object; use Java.Lang.Object;
    use Java.Lang;

package C is
    type Typ is new Java.Lang.Object.Typ with record
        Field : Integer;
    end record;
    type Ref is access all Typ’Class;

    function new_C (This : Ref := null) return Ref;
    function new_C (I : Java.int; This : Ref := null) return Ref;
    function new_C (I, J : Java.int; This : Ref := null) return Ref;
    pragma Java_Constructor (new_C);
end C;
package body C is

  function new_C (This : Ref := null) return Ref is
    Super : Java.Lang.Object.Ref := new_Object (Object.Ref (This));
  begin
    This.Field := 3;
    return This;
  end new_C;

  function new_C (I : Java.int; This : Ref := null) return Ref is
    Super : Java.Lang.Object.Ref := new_Object (Object.Ref (This));
  begin
    This.Field := I;
    return This;
  end new_C;

  function new_C (I, J : Java.int; This : Ref := null) return Ref is
    Ignore : Ref := new_C (I + J, This);
  begin
    return This;
  end new_C;
with type Java.Lang.String.Ref is access;

package Java.Lang.Object is
  type Typ is tagged limited null record;
  type Ref is access all Typ’Class;
end Java.Lang.Object;
type Typ is tagged limited null record;

✔ Java classes are limited types
  - no object assignment (no deep copy)

✔ Warning:
  - must call a constructor to create an object (Java semantics)
  - No stand-alone Java object can be created on the Ada side if there is no default constructor that the compiler can call
Problem if no default constructor

```java
public class Bar {
    public Bar(int i) {...} // only constructor
    ...
}
```

Obj : Bar.Typ; – ???
JGNAT: Walk through a simple demo applet
Applet vs Application

✔ Applet: Embedded in a Web browser.
  – Security restrictions

✔ Application: Independent program
  – Full access to the Java API
Applets (1)

✔ Extends the java.applet.Applet class
✔ Embedded in an HTML document through the <Applet> tag.
✔ Four special methods:
  - **Init:** called by the browser to inform this applet that it has been loaded into the system
  - **Start:** called by the browser to inform this applet that it should start its execution
  - **Stop:** called by the browser to inform this applet that it should stop its execution
  - **Destroy:** called by the browser when the applet is being reclaimed and should destroy any resources that it has allocated
Applets (2)

✓ Security restrictions:
  - No access to the file system on the client
  - No access to serial or parallel ports
  - And others...

✓ Full access to GUI
The Example
JGNAT: Complete Ada Solution
100% Java compatible