Reverse conversion from Intel JXE format to Oracle JAR

The used JVM is J9[1], development tools for technology are IBM WebSphere Everyplace Micro Environment and WebSphere Studio Device Developer.

JAR to JXE conversion is done with jar2jxe utility. For reference, a package for Linux-based OS contains this utility - ibm-ws-rt-jre-1.0-1.0-linux-i386.tgz. Library ibm-ws-rt-i386-1.0-1.0\jre\bin\realtime\libj9jextract.so is included in the archive and have linked j9util.lib linked to it. This library implement j9bcutil_dumpRomClass and j9bcutil_dumpRomMethod functions.

JXE Format

JXE is optimized version of JAR, which speeds up JVM performance. All classes from classic JAR are gathered into one file rom.classes, which has structure of J9ROMImage (see Table 1).

Table 1. J9ROMImage sctructure

Offset	Name	Type	Comment
0	signature	uint32	"J99J"
4	flags_and_version	uint32	?
8	rom_size	uint32	rom.classes size
12	class_counter	uint32	J9ROMClass count
16	jxe_pointer	relative	?
20	toc_pointer	relative	pointer to array
24	first_class_pointer	relative	?
28	aot_pointer	relative	?
32	symbol_file_id	byte[0x10]	?

And type relative is int32, but this value should be added to offset of this value in file to retrieve final offset of element in file.

Using field toc_pointer can be found array of class descriptors:

Table 2. Elements in array of J9ROMClass descriptors

Offset	Name	Type	Comment
0	class_name	string_relative	Pointer to class name
4	class_pointer	relative	Pointer J9ROMClass

Ant type string_relative similar to relative type, but result offset point to UTFString object:

Table 3. UTFString

[1] http://wiki.eclipse.org/J9

Offset	Name	Type	Comment
0	length	word	
2+	value	byte[]	value of the string

So parsing descriptors toc_pointer all J9ROMClass can be found JXE:

Table 4. J9ROMClass

Offset	Name	Type	Comment
0	size	uint32	
4	single_scalar_static_count	uint32	
8	class_name	string_relative	
12	superclass_name	string_relative	name of base class
16	access_flags	uint32	similar to oracle acces_flags for classes (need & 0xFFFF)
20	interface_count	uint32	
24	interface_pointer	relative	pointer to array J9ROMInterface
28	method_count	uint32	
32	method_pointer	relative	pointer to array J9ROMMethod
36	field_count	uint32	
40	field_pointer	relative	pointer to array of J9ROMField
44	object_static_count	uint32	
48	double_scalar_static_count	uint32	
52	rom_constant_pool_count	uint32	
56	ram_constant_pool_count	uint32	
60	crc	uint32	
64	instance_size	uint32	
68	instance_shape	uint32	
72	cp_shape_description_pointer	relative	
76	outer_class_name	string_relative	
80	member_access_flags	uint32	
84	inner_class_count	uint32	
88	inner_class_pointer	relative	
92	major	word	

Offset	Name	Type	Comment
94	minor	word	
98	optional_flags	uint32	used to define existed attributes for class
102	optional_pointer	relative	base address of defined attributes for class

Using field interface_count and interface_pointer of J9ROMClass all interfaces of class can be parsed:

Table 5. J9ROMInterface

Offset	Name	Type	Comment
0	name	string_relative	

Using field field_count and field_pointer of J9ROMClass all fields of class can be parsed:

Table 6. J9ROMField

Offset	Name	Type	Comment
0	name	string_relative	
4	signature	string_relative	type of field can be defined
8	access_flags	uint32	similar to oracle (& 0xFFFF)
12	unknown1	uint32	access_flags & 0x400000
16	unknown2	uint32	access_flags & 0x400000 and access_flags & 0x40000
20	unknown3	uint32	access_flags & 0x40000000

Using field method_count and method_pointer of J9ROMClass all methods of class can be parsed:

Table 7. J9ROMMethod

Offset	Name	Type	Comment
0	name	string_relative	
4	signature	string_relative	
8	modifier	uint32	if modifier & 0x100 then native method (after max_stack arg_count(u8), temp_count(u8), stub(u8) and native_arg_count(u8)
12	max_stack	word	
16	bytecode_size_low	word	

Offset	Name	Type	Comment
18	bytecode_size_high	byte	
19	arg_count	byte	
20	temp_count	word	
22	bytecode	byte[]	
22 + ?	caught_exception_count	word	In case of modifier & 0x00020000
24 + ?	thrown_exception_count	word	In case of modifier & 0x00020000

Bytecode of method follow object J9ROMMethod, the size of bytecode depend on modifier:

Listing 1. Calculating bytecode size

```
bytecode_size = bytecode_size_low
if modifier & 0x00008000:
    bytecode_size += bytecode_size_high << 16
bytecode_size *= 4
if modifier & 0x02000000:
    bytecode_size += 4
```

Field modifier (& 0xFFFF) correspond to access_flag in Oracle JAVA specification[2].

If modifier & 0x00020000 then array of J9ROMCatchException followed bytecode of method. This array describe exceptions with handled in this method, the number of elements in array equal to value of field caught_exception_count in J9ROMMethod.

Table 8. J9ROMCatchException

Offset	Name	Type
0	start	uint32
4	end	uint32
8	handler	uint32
12	catch_type	uint32

Array of J9ROMThrowException follow array of J9ROMCatchException (or bytecode of method), the number of elements in array equal to value of field thrown_exception_count in J9ROMMethod.:

Table 9. J9ROMThrowException

Offset	Name	Type
0	throw_type	uint32

JXE constant pool

[2] https://jcp.org/aboutJava/communityprocess/maintenance/jsr924/JVMS-SE5.0-Ch4-ClassFile.pdf

Constant pool in Intel JXE diff a lot from Oracle JAVA and is optimized version. So element in constant pool have structure:

Table 10. Structure of elements in constant pool

Offset	Name	Type
0	type	uint32
4	value	uint32

Field type really mean type only in three cases:

0 – Int or Fload type, field value – is real value

1 – String type, field value is a relative offset to UTFString

2 – Class type, field value is a relative offset to UTFString, which is a class name.

In other cases field type is part of value and constant can be one of reference-like or Long (or Double) type. To reference-like type include InterfaceRef, FieldRef and MethodRef. In Long (or Double) constants fields type and value are real value. In reference-like constants field type is offset to Class name and field value is offset to Name and Descriptor of method(or field).

Also in Oracle Java classes constant pool contain stub (null) constants, which follow Double or Long constants. This stub (null) constants change reference indexes to constants from bytecode.

So for conversion constant pool should be rebuild with actions:

1. Restore the real type of constant
2. Expand optimized constant (reference-like constants and Class constants) to simple constant from Oracle. So:
    a. Class constant should be transform to CONSTANT_Class_info[3] and CONSTANT_Utf8_info[4]
    b. String constant should be transform to CONSTANT_String_info[5] and CONSTANT_Utf8_info
    c. Reference-like constants should be transform to (CONSTANT_Fieldref_info, CONSTANT_Methodref_info, and CONSTANT_InterfaceMethodref_info [6]), CONSTANT_Class_info, CONSTANT_NameAndType_info[7]
3. Add stub constants in constant pool after Long or Double constants.

To define is constant is reference-like its value of fields can be checked as offsets in file, if error happened in this process then constant is Long (or Double). Correct type of constant can be defined only during parsing opcodes of bytecode.

JXE bytecode

Bytecode in Intel JXE differ from Oracle JAR file in:

[3] https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-4.html#jvms-4.4.1
[4] https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-4.html#jvms-4.4.7
[5] https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-4.html#jvms-4.4.3
[6] https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-4.html#jvms-4.4.2
[7] https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-4.html#jvms-4.4.6

1. Byte order of index constant in constant pool, so all opcodes which operates with constant should be transformed:

Table 11

JBgetstatic JBputstatic JBgetfield JBputfield JBinvokevirtual JBinvokespecial JBinvokestatic JBnew JBificmple JBifacmpeq JBificmpge	JBldcw JBldc2lw JBldc2dw JBsipush JBifeq JBifne JBiflt JBifge JBifgt JBifle JBificmpeq	JBinvokeinterface2 JBldc JBtableswitch JBlookupswitch JBmultianewarray JBgotow JBjsr JBifnull JBifnonnull JBificmpgt JBgoto	JBanewarray JBcheckcast JBinstanceof JBificmpne JBificmplt JBifgt JBifle JBificmpeq JBificmple JBifacmpeq JBifacmpne

2. In Intel-version there are some opcode, which absent in Oracle-version:
    a. JBaload0getfield -> JBaload0
    b. JBreturn0 and JBsyncReturn0 -> return (0xb1)
    c. JBreturn1 and JBsyncReturn1 -> return (0xb0)
    d. JBinvokeinterface2 -> JBinvokeinterface + JBnop + JBnop
3. In bytecode reference to constant should correlate to rebuild constant pool, especially after adding stub constants.

Tool for JXE to JAR conversion

It was shown that conversion from JAR to JXE leaves sufficient information to restore Oracle class files, the only challenge is creation of constant pool and bytecode modification. Proof of concept tool is in supplementray materials of this report. The result of this tool is generated JAR file, which can be decompiled using known versions. As a result of possible inaccuracies in definition of constant type, some well known decompilers (such as jd[8]) failed. So the best tool in this case is CFR decompiler[9].

[8] http://jd.benow.ca/

[9] http://www.benf.org/other/cfr/